Reinforcement Learning based Control for Pouring Liquids

Weitung Chen
Massachusetts Institute of Technology
weitung@mit.edu

Ngoc La
Massachusetts Institute of Technology
ntmla@mit.edu

***Abstract*—Humans can pour liquid easily from one container to another. In this project, we aim to exploit the dynamics of controlling the simple pouring action to minimize the spillage and the time to finish the pour. To simplify the fluid dynamics in the container, we have built a simulation environment that simplifies the liquid to many small spheres. The simulation environment takes account of the dynamics of these spheres colliding with each other and the container. We perform reinforcement learning in the simulation environment and apply the policy gradient algorithm to find the best control of the pouring action. The proposed algorithm shows the trend to getting higher rewards in 100 episodes. In the future, we aim to apply this method of finding optimal pouring on a robot that helps tofu's food production process, which requires pouring high-temperature soymilk from one bucket to another.**

***Index Terms*—pouring, robotics, Reinforcement Learning, tofu**

I. Introduction

This project aims to explore the manipulation of liquid through the application of robot pouring action. In this problem, the only control input would be the torque of the rotating bucket filled with liquid. We aim to find the optimal control input that minimize the spillage when pouring from one bucket to another. Fluid, in general, are difficult to model for planning purposes, and therefore a data-driven approach is considered for robot manipulation. With data obtained from the simulation environment, one can train machine learning models of the system to predict the fluid behavior and subsequently plan torque commands to achieve minimal spills.

The motivation behind this project comes from the food production process of tofu. Traditional process of making tofu requires pouring a bucket (usually more than 10 liters) of soymilk at around 200 degree Fahrenheit to another bucket with coagulant as quickly as possible. The goal of this project is to develop a mechanism with an optimal control algorithm to automate this pouring process to reduce the risk of handling a bucket of liquid with such a high temperature.

In this project, the problem formulation is similar to the problem in Huang et al.'s [1] paper. More detailed problem definition is written in the following section. This is an underactuated control problem as we only have direct control to one degree of freedom of the rotating bucket, which contains liquid that we are unable to directly manipulate. As the bucket is rotating, the moment of inertia may change, which requires an adjustment to the control. This interaction and the dynamics are hard to model and predict. Therefore, a reinforcement learning approach is considered in this case.

There are several studies that try to tackle the problem of robot pouring liquid. Pan et al. [2] studied the control of pouring based on optimization methods while Huang et al. [1] applied Self-Supervised Learning, more specifically LSTM, to train robots on the pouring action. Model-free reinforcement learning methods have been used widely in training robots to perform tasks that are hard to model. In this project, we investigate into the performance of policy gradient algorithm on manipulating liquid pouring, aiming to find optimal control to minimize the spillage.

II. Problem Formulation

The problem is formulated as follow: There is one rotating bucket originally filled with small spheres centering at $(x_{rotating}, y_{rotating}, z_{rotating})$ and a target bucket centering at $(x_{target}, y_{target}, z_{target})$. The size of the bucket container is the same as one of the most common dimension of the bucket used in the field of tofu making process.

$$\begin{aligned} bucketDiameter &= 24cm \\ bucketHeight &= 33cm \\ bucketThickness &= 0.5cm \end{aligned}$$

The liquid is formulated as many small spheres, or molecules. For this project, we have initialized 150 spheres randomly in the rotating bucket at $t = 0$.

At step t, the robot is given an observation of its state $\mathbf{x}$. There are 5 state variables $(\theta, \dot{\theta}, V_{rotating}, V_{target}, V_{spill})$, which corresponds to rotate angle, angular velocity, volume of liquid in the rotating bucket, volume of liquid in the target bucket, and the volume of liquid spilling out. The state space of these variables are $(0, 1)$, which is normalized to this range. Initially when $t = 0$, θ, $\dot{\theta}$ are set to 0.

Afterwards, the robot plans to apply a torque A_t on the rotating bucket. The action space is $(-1, 1)$ and is discrete. We have tried different granularities of the discrete actions: equally dividing the action spaces to 2 or 11 actions.

For each step, a reward is the weighted sum of the volume of liquid in the target bucket and the volume of liquid spilled

after applying the torque implied by action a_t. The received reward is calculated by the following equation:

$$r(\mathbf{x}_t, a_t) = K_{target} * V_{target,t+1} - K_{spill} * V_{spill,t+1} \quad (1)$$

where the Ks are the weighting factor of each of the liquid volumes to the reward. Also note that we have set a time constraint for the agent to perform the pouring action, if the simulation world time exceeds 10 seconds, equivalent to 10,000 steps, current episode will end and the cumulative reward $R(\tau)$ will be recorded. The cumulative reward $R(\tau)$ after τ steps with a discount factor γ is defined as the following equation:

$$R(\tau) = \sum_{t=0}^{\tau-1} r * \gamma^t \quad (2)$$

III. Methods

A. Simulation

In this problem, the liquid is modeled as a bunch of spheres (molecules). We build the simulation from scratch to handle the dynamics of the liquid molecules and their collisions with each other, the ground, and the buckets. The simulator is visualized with VPython. Figure 1 demonstrates the simulation environment. At each time step in the simulation, we update the states of the spheres and check the collisions of them with each other, with the ground, with the buckets. For simplicity, gravitational force is the only force acting on the spheres and that the collision with the rotation bucket is calculated in the rotating bucket frame (when the liquid molecules are inside the rotating bucket).

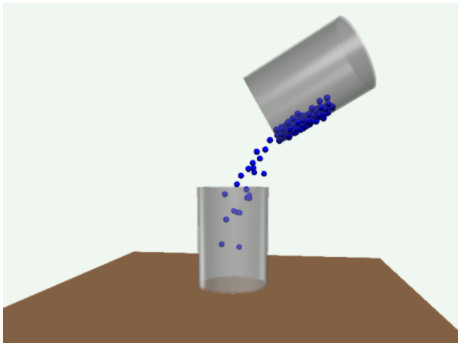

Fig. 1. VPython render of the simulation

1) Collision between liquid molecules: The general formula for collision between 2 water molecules (same mass) is described in the equation (3) and (4). The parameter e is the elasticity parameter, which is 0 for completely inelastic collision. In our simulation, we tested with various values of e and found that e = 0.5 works reasonably.

$$\vec{v_1'} = \vec{v_1} + \frac{1}{2}(v_{2x} - v_{1x})(1+e)\vec{i} - \frac{1}{2}(v_{2y} - v_{1y})e\vec{j} \quad (3)$$

$$\vec{v_2'} = \vec{v_2} + \frac{1}{2}(v_{1x} - v_{2x})(1+e)\vec{i} + \frac{1}{2}(v_{2y} - v_{1y})e\vec{j} \quad (4)$$

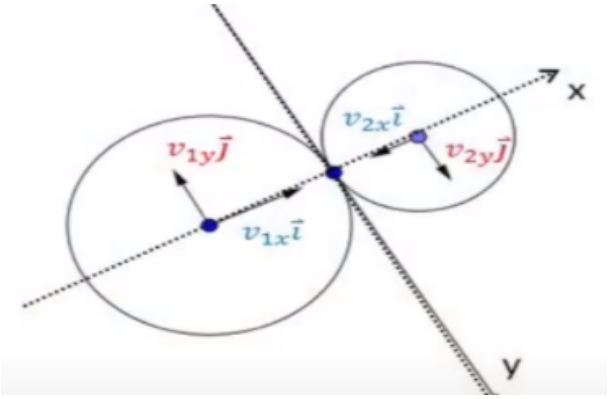

Fig. 2. Collision of two molecules

2) Transformation between frames: To model the collision between liquid molecules and the rotating container, we need to transform the position, velocity, and acceleration of each molecule from the ground coordinate system into the bucket coordinate system. These two frames are different by a translation of $\vec{x}_{rotating} = (x_{rotating}, y_{rotating})$ of the center of rotating bucket, and rotation of the angle theta (defined as an angle from y-axis of the global frame to y-axis of the bucket frame), angular velocity $\dot{\theta}$, and angular acceleration $\ddot{\theta}$. Note that the z direction is similar to both coordinate systems, thus, we only consider 2D transformation for simplicity. The transformation between global (O) and bucket (B) coordinate is as follow:

$$R_{OB} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \quad (5)$$

$$\vec{x_B} = R_{OB}(\vec{x} - \vec{x}_{rotating}) \quad (6)$$

$$\vec{v_B} = R_{OB}\vec{v} + \dot{R}_{OB}(\vec{x} - \vec{x}_{rotating}) \quad (7)$$

$$\vec{a_B} = R_{OB}\vec{a} + 2\dot{R}_{OB}\vec{v} + \ddot{R}_{OB}(\vec{x} - \vec{x}_{rotating}) \quad (8)$$

3) Collision between the molecule with surfaces: The collision between the molecule with the bucket's wall, bottom, and the ground are modeled similarly as a collision of the molecule with a surface. In the direction perpendicular to the surface, the velocity direction changes to the opposite sign, and the magnitude changes by a factor of e in range of [0,1]. Note that when e = 1, the collision is completely elastic. In our model, we chose e = 0.1, which well represents the stickiness (non-completely elasticity) of the liquid molecule when it collides with a surface. In the direction tangent to the surface, the velocity does not change either in direction nor in magnitude.

$$v_{\perp}^{'} = -v_{\perp} \times e_{surfaces} \quad (9)$$

$$v_{\parallel}^{'} = v_{\parallel} \quad (10)$$

4) Controller dynamics: The controller in the problem is the torque applied at the center of the rotating bucket. We have:

$$Torque = I * \ddot{\theta} \quad (11)$$

$$\dot{\theta} \longleftarrow \dot{\theta} + \ddot{\theta} * dt \tag{12}$$

$$\theta \longleftarrow \theta + \dot{\theta} * dt \tag{13}$$

where I is the moment of inertia of the bucket and liquid inside, which can be calculated by adding all the moments of inertia of all molecules inside the bucket and the bucket's moment of inertia. $\ddot{\theta}$ is the angular acceleration. Using equation (11), we can update $\ddot{\theta}$, and thus angular velocity of the rotating bucket ($\dot{\theta}$), and thus angle of the rotating bucket (θ) in each cycle of time t.

B. Model-free Policy Search

In this problem, we have to keep track of many molecules and their collisions, therefore, the model-free policy search using Reinforcement learning method would be the best approach for this black-box optimization problem. The black-box in this problem is our simulation, which takes in the action a_{t-1} at step (t-1), in this case is the torque applied to the rotating bucket, and returns the updated state, which is defined as $\mathbf{x}_t = [\theta_t, \dot{\theta}_t, V_{rotating,t}, V_{target,t}, V_{spill,t}]$ where θ_t and $\dot{\theta}_t$ is the angle and angular velocity of the rotating bucket at step t. The cumulative reward after τ steps with a discount factor γ is as the equation (14).

$$R(\tau) = \sum_{t=0}^{\tau-1} \gamma^t * r(\mathbf{x}_t, a_t) \tag{14}$$

The goal is to maximize the expected reward:

$$\max_{w} \mathbb{E}_{\pi_w}[R(\tau)] \tag{15}$$

where π_w is the policy, in this problem, we use $softmax(\mathbf{x}_t \cdot w)$ to be the probabilities of choosing action ($\mathbf{x}$ is the state we observed after each step).

$$p_w(\mathbf{x}_t) = softmax(\mathbf{x}_t \cdot w) \tag{16}$$

Therefore, to solve the problem, we need to find the best parameter w of the policy that maximizes the expected reward. Applying gradient descent, w is updated in the direction of the gradient with the learning rate α after each training episode as following:

$$w \longleftarrow w + \alpha \nabla_w \mathbb{E}_{\pi_w}[R(\tau)] \tag{17}$$

According to the derivation in chapter 20, part 20.1.1 of the Underactuated Robotics textbook [3], we have

$$\nabla_w \mathbb{E}_{\pi_w}[R(\tau)] = \mathbb{E}_{\pi_w}[R(\tau) \nabla_w log(p_w(\mathbf{x}_t))] \tag{18}$$

Combining with the long form of expectation, equation (17) can be written as:

$$w \longleftarrow w + \alpha \sum_{t=0}^{\tau-1} R(\tau) \nabla_w log(p_w(\mathbf{x}_t)) \tag{19}$$

IV. Results

We have applied model-free policy search and perform learning in the simulation environment. Due to time constraints, we ran 100 episodes in most of our training. The experiment is conducted first using binary actions and then with more discretized actions to test with a more complex control.

A. Binary actions

We started with the binary action controller, meaning the action domain is -1,1 and thus the torque is either -τ_{max} or τ_{max}. Figure 3 shows slight improve in the rewards within 100 episodes. We only run 100 episodes per trial because the simulation runs very slow as it needs to calculate all the collisions between molecules and container. Although the number of episodes is not enough to conclude anything about the convergence, we can see that there's a trend of getting higher rewards.

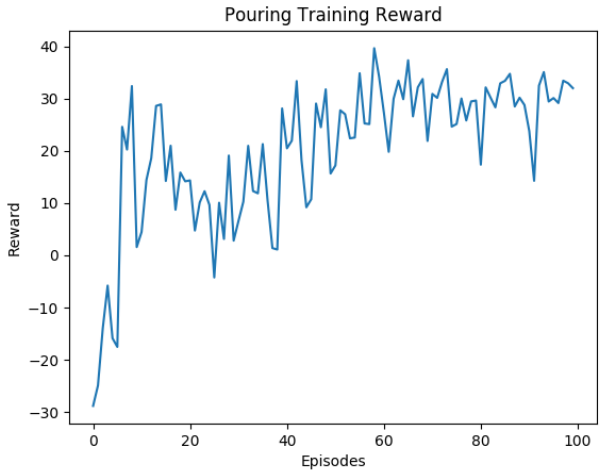

Fig. 3. Rewards of 100 episode training of multiple discrete actions

Figure 4 shows the angle evolution of the best-reward episode. From this we can see that the angle increase with a constant rate before capping at the maximum angle. This is likely to be a suboptimal policy as the hard theta stop might still leads to a decent amount of liquid spillage.

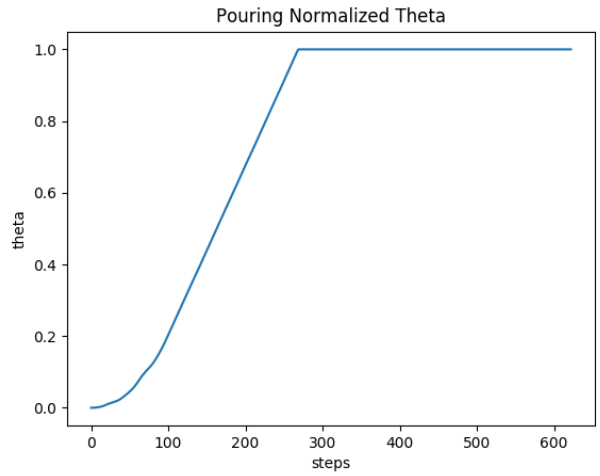

Fig. 4. Angle of the rotating bucket in the best trajectory case of the binary action controller

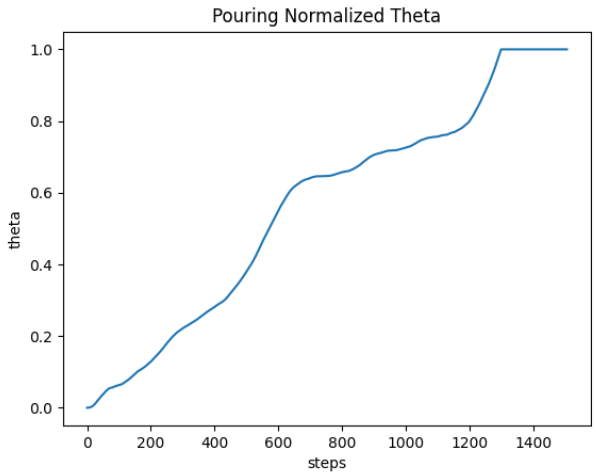

Fig. 6. Normalized angle of the rotating bucket in the best trajectory case of the multiple discrete action controller

B. Multiple discrete actions

We also experimented with a more complicated case where we discretized our action space to more actions. In this experiment, we have splitted our action to 11 actions. Figure 5 shows the reward evolution of this training. It implies that the training didn't show signs of convergence in 500 episodes. Measures that can improve this training includes: increasing the training episodes, use deterministic policy to enhance stability, adjust the gradient update method to best-suit the multi-action scheme. The above are some of the ways that we can do in the future to try improve the policy.

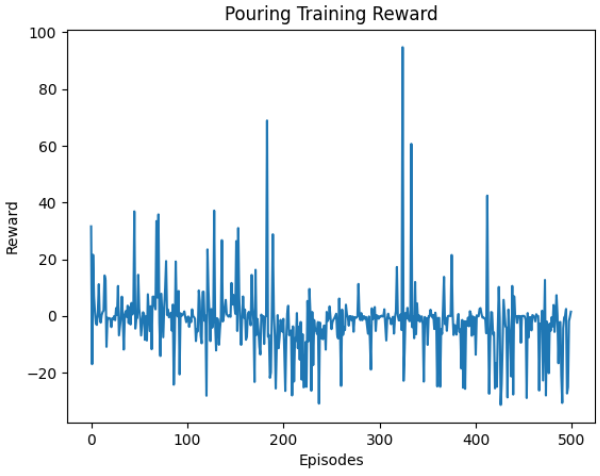

Fig. 5. Rewards of 500 episode training of multiple discrete action controller

Figure 6 also shows the angle evolution of the best-reward episode for the multi-discrete actions training. Although this might be the result of random exploring, the angle revolution looks less likely to spill as the motor slows down during the pouring period. The video for this best-reward episode simulation can be found here: https://youtu.be/0pMqOk9CDgw.

One thing that we planned but haven't have time to do was applying the technique on the real experimental setup as shown in Figure 7. We think this is interesting as we can get to see the difference between simulation and the real robot pouring. This is definitely something that we can investigate into in the future.

V. Conclusions and Future Works

In conclusion, we have implemented a simulation environment based on VPython. The fluid behavior is represented by 150 small spheres, taking account for the collision between them and the containers. The simulation environment gave us a better understanding of how the liquid will behave under various control inputs and states.

Based on the simulation, we perform reinforcement learning and applied policy gradient algorithm on the agent. We started with binary action control with either -1, or 1. The results show that it is exploring different control inputs in the first 100 episodes and it shows a tendency to get a better reward. We also tried applying discrete action inputs and trained for 500 episodes. The training result shows that there are still some issues to overcome. However, this is a start to finding more complex control to manipulate the liquid and perform the pouring action. In the future, we hope that this can be applied on a real robot and let the robot gain accurate liquid pouring skill.

References

[1] Yongqiang Huang, Juan Wilches, and Yu Sun. (2020). Robot Gaining Accurate Pouring Skills through Self-Supervised Learning and Generalization.

[2] Zherong Pan, Chonhyon Park, and Dinesh Manocha. 2016. Robot motion planning for pouring liquids. In Proceedings of the Twenty-Sixth International Conference on International Conference on Automated Planning and Scheduling (ICAPS'16). AAAI Press, 518–526.

Fig. 7. Experimental setup for robot pouring liquid.

[3] Russ Tedrake. Underactuated Robotics: Algorithms for Walking, Running, Swimming, Flying, and Manipulation (Course Notes for MIT 6.832). Downloaded on May 20, 2021 from http://underactuated.mit.edu/