HACMan-AFF: Learning Object Affordances for Rolling, Spinning, and Sliding Manipulation

Jaskrit Singh
School of Robotics Engineering
Worcester Polytechnic Institute
Worcester, Massachusetts
jsingh3@wpi.edu

I. Introduction

Non-prehensile manipulation is an important open research question in robotics. While humans can precisely control objects through rolling, spinning, and sliding, robots struggle to perform these actions. There have been attempts to recreate these abilities in robots, but they all fall short.

One approach is to use a reinforcement learning (RL) algorithm to learn a policy that takes in a start and a goal position and outputs an action. This strategy has been used successfully [1], but the strategy has the limitation that it creates a reactive policy rather than making a deliberate plan. This can be an issue because if the policy fails, there is no way to switch to a different strategy without retraining the model.

A different approach would be to model the dynamics of the object and then use a motion planner to create a plan to reach the goal. Unlike the RL approach, this approach could be used for long-horizon planning and for avoiding obstacles. Unfortunately, the dynamics of non-prehensile manipulation are complicated, so it is not feasible to create a precise dynamical model.

One way to simplify the dynamics of non-prehensile manipulation is to break down manipulation into categories such as rolling, spinning, and sliding. If we could make an object exhibit just one of these motions, it would be easier to predict how the object will behave.

This paper will isolate three non-prehensile manipulation behaviors: pushing straight, rolling, and spinning. We will use RL to train a model that attempts to do one of these actions while avoiding the others.

We define pushing straight as translating an object in the plane of the table, while not affecting its orientation. When pushing straight, the object should slide, but it should not roll or spin.

We define the roll action as rotating an object in an axis perpendicular to the z-axis. Rolling does not involve sliding or spinning about the z-axis.

Finally, we define the spin action as rotating an object in the Z-axis. It is difficult to only spin an object when performing non-prehensile manipulation with a single gripper, so some sliding will be allowed for the spin action. However, the object should not roll during the spin action.

Training models that perform each of these actions will allow us to create the beginning of a model for how objects react to non-prehensile manipulation. This includes an understanding of which actions can be used in each of the different resting states of an object, as well as how stable each action is. These models could later be incorporated into motion planning algorithms, thereby combining the strengths of reinforcement learning and planning.

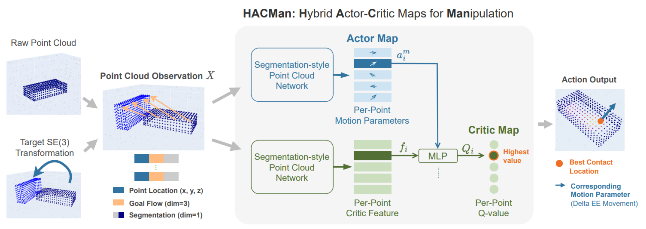
HACMan: Hybrid Actor-Critic Maps for Manipulation
Raw Point Cloud
Target SE(3) Transformation
Point Cloud Observation X
Point Location (x, y, z)
Goal Flow (dim=3)
Segmentation (dim=1)
Actor Map
Segmentation-style Point Cloud Network
a_i^m
Per-Point Motion Parameters
Critic Map
Segmentation-style Point Cloud Network
f_i
MLP
Q_i
Highest value
Per-Point Critic Feature
Per-Point Q-value
Action Output
Best Contact Location
Corresponding Motion Parameter (Delta EE Movement)

Fig. 1: HACMan Architecture

II. Related Work

This paper is based on Hybrid Actor-Critic Maps for Manipulation (HACMan) [1]. HACMan uses RL and Q-learning to develop a model that can use non-prehensile manipulation to move an object to a given goal. HACMan selects the best contact point on the object and selects the best motion to make from that contact point.

The HACMan architecture can be seen in Figure 1. First, three depth cameras generate a point cloud of the object. HACMan then appends a goal flow vector to each point, showing where that point must go to get to the goal. The augmented point cloud is then fed to the actor and critic map neural networks. The actor map decides what motion to take for each contact point in the point cloud. Meanwhile, the critic map gives an initial estimate of how valuable it thinks it is to choose a certain contact point. Finally, both the actor and critic map outputs are fed to a multi-layer perceptron, which gives a final estimate for the value of choosing each contact point. HACMan uses an exploration policy and Q-learning to develop these networks. Once the networks are sufficiently trained, HACMan can be evaluated by setting it to always take the maximum value action.

HACMan is able to consistently complete non-prehensile manipulation tasks up to a precision of 3 cm in a simple environment. It achieved an 85% success rate in simulation and a 50% success rate in the real world (with no real-world training).

Although HACMan was able to achieve high-level performance, it has a few key limitations. First, HACMan is incredibly data and RAM-hungry to train. After training for 17,000 iterations over 8 hours with 100 GB of RAM, we only see a 10% success rate, and documentation suggests that the full model requires 200,000 iterations to train to the 85% training success rate achieved in the paper.

Furthermore, while HACMan was able to generalize to the real world from simulated training, there is still a noticeable gap that will likely require real-world training to address. The fact that HACMan requires so many examples suggests that fine-tuning the model in the real world could be time-consuming.

Another limitation of HACMan is its limited ability to choose a different plan if the initial plan does not work. Although HACMan evaluates multiple contact locations and estimates of the value of each of these locations, it does not have a way to select between them. This leads to failure cases where it can try and fail to do the same action over and over again, rather than trying a different strategy.

HACMan is also limited by its inability to navigate around obstacles. HACMan was trained and evaluated in a clear environment, so when an obstacle is added to the environment, its performance drops. It would likely require combining HACMan with a motion planner to overcome this issue.

Overall, HACMan shows strong non-prehensile manipulation ability, but it would benefit from being combined with a more deliberate, planning-based approach.

III. Methodology

This paper focuses on simplifying HACMan to create simpler models that would be easier to combine with planning-based approaches. The main simplification was to remove the goal-flow vectors from the neural network. This allows us to give the model a general reward function to minimize instead of requiring a goal location to be specified. Three loss functions were generated to incentivize our desired actions: push straight, flip, and spin.

A. Push Straight

To train the model to push straight, we incentivize translational motion up to a distance of 5 cm, and we penalize rotation. We saturate the position loss at 5 cm so that the model focuses on making small, stable pushes rather than trying to make longer, potentially unstable ones. Given a translation of $\mathbf{p}$ (in meters) and and an angle of rotation θ (in degrees), we can define our loss function as follows:

$$\mathcal{L}_{pos} = 20 \times max\left(0.05 - \|\mathbf{p}\|, 0\right)$$

$$\mathcal{L}_{rot} = \frac{\theta}{180^\circ}$$

$$\mathcal{L}_{pushStraight} = \mathcal{L}_{pos} + \mathcal{L}_{rot}$$

We consider a push straight to be a success if the push was at least 5 cm long and the object orientation changed by less than 5°.

B. Roll

The roll action was primarily designed for changing which face an object is resting on, for example changing the face of a die. However, it is difficult to decouple this action from the action of say rolling a ball. For this reason both of these actions were combined into a single action. As before, we design a loss function to incentivize these behaviors.

The first part of the loss function, $\mathcal{L}_{rotAxis}$, is designed to make sure rotation occurs along the correct axis. We define it with the formula $\mathcal{L}_{rotAxis} = |\omega \cdot \hat{z}|$, where ω is the unit vector axis of rotation and $\hat{z}$ is a unit vector normal to the table surface.

The second part of the loss function is $\mathcal{L}_{rotAngle}$. We want to encourage large rotations of up to 90°, but since a cube rotated 60° would rotate the last 30° on its own, we will set a smaller goal of only 60°. Thus we set $\mathcal{L}_{rotAngle} = max(1 - \frac{\theta}{60^\circ}, 0)$.

Although rolling includes translational motion, we want to discourage excess sliding, so we put a small loss on movement in the XY-plane, $\mathcal{L}_{pos} = \frac{1}{2}\|\mathbf{p}_{xy}\|$. This corresponds to a loss of 0.05 for a push of 10 cm.

Our roll loss is $\mathcal{L}_{roll} = \mathcal{L}_{rotAxis} + \mathcal{L}_{rotAngle} + \mathcal{L}_{pos}$. A roll action is successful if it causes a rotation of at least 60° with an axis within 45° of being flat.

C. Spin

The spin action involves rotating an object in the Z-axis. This is difficult to perform for most objects without introducing some translational motion as well, so we allow for some leeway in the translational portion of the loss.

We can use the same basic framework that we used for the roll loss $\mathcal{L}_{spin} = \mathcal{L}_{rotAxis} + \mathcal{L}_{rotAngle} + \mathcal{L}_{pos}$. However, we need to make some changes to each of the terms. To incentivize a rotation axis close to the Z-axis, we set $\mathcal{L}_{rotAxis} = 1 - |\omega \cdot \hat{z}|$. Next, since rotation in the Z-axis is difficult with one gripper, we only aim for a 30° turn. This means $\mathcal{L}_{rotAngle} = max(1 - \frac{\theta}{30^\circ}, 0)$. Lastly, we want to be stricter with translational motion for spinning than we were for rolling because we want the robot to turn objects with as tight of a turning radius as possible. We set $\mathcal{L}_{pos} = \frac{5}{2}\|\mathbf{p}_{xy}\|$, which corresponds to a loss of 0.25 for a 10 cm push. Finally, we define success as rotating at least 30° at an angle within 45° of vertical.

IV. Results

A. Platform

The three models were trained on the WPI Turing Cluster with 100 GB of RAM, a GPU, and a 10 hour time limit. Unfortunately, roll training crashed after only 2 hours and spin training crashed after 5 hours. These crashes were likely due to running out of RAM during an evaluation step or due to a memory leak. Fortunately, despite the crashes, the

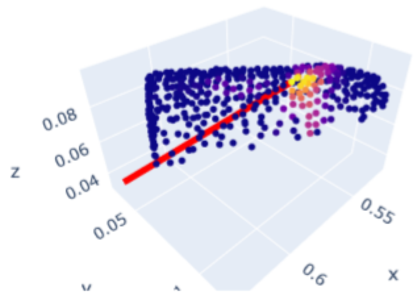

Fig. 2: Critic Map for the push straight action along with the action for the highest value point

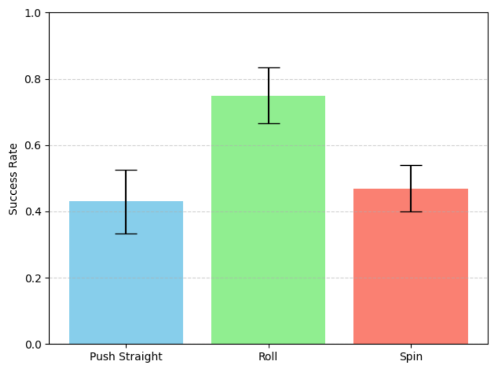

Fig. 3: Mean Action Success Rate for 200 trials

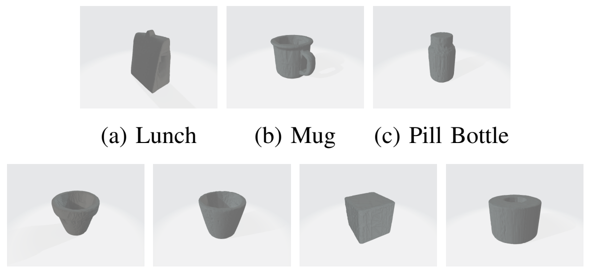

Fig. 4: Seven Objects Chosen for Further Testing

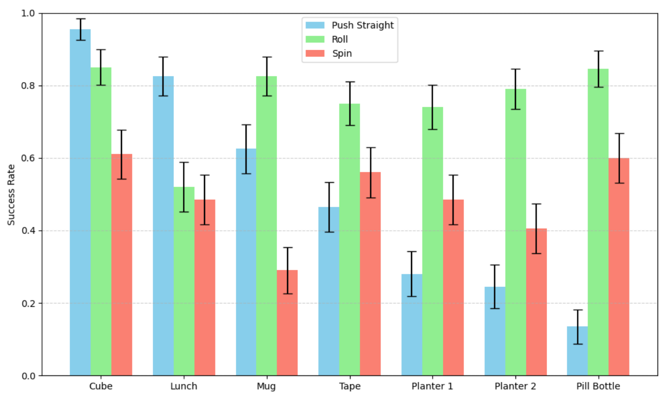

Fig. 5: Success rates for performing each action for each selected object

models produced meaningful results, so the models were not re-trained. Code implementation can be found at www.github.com/JazKarit/hacman.

B. *Evaluation with Random Objects*

Each model was tested for 200 stable starting positions for the diverse set of household objects from [1]. The critic map for the trained push straight action is shown in Figure 2 as an example of the output of the models. We can see that, for this object, the model learned to push from the top and along the length of the object.

The success rates of the trained models are shown in Figure 3. The results show that the model learned to perform the roll action very well with a success rate of 75%. The 43% success rate for the push straight model is also good considering the small tolerance of only 5°. The spin model performed the worst. It was a relatively simple task with high tolerances, and it only achieved a success rate of 47%. This suggests that spinning is a difficult task for a single manipulator.

C. *Evaluation with Chosen Objects*

We evaluate the models on a chosen set of objects to see how individual differences in the object geometries affect performance. These objects are shown in Figure 4. Looking at results in Figure 5, we can see that the success rate can heavily depend on the object. This is especially true for the push straight action, which ranges from a 14% success rate with the pill bottle to a 96% success rate with the cube.

All of the actions seem to perform best for the cube, but for the other objects, some actions are harder than others. Many of the objects are roughly cylindrical, and it seems like the closer an object is to a narrow cylinder, the harder it is to push straight. Looking at the cylindrical objects in the order of success rate, we can see that the mug has a handle, the tape is a wide cylinder, the planters are narrower frustums, and the pill bottle is the narrowest and is roughly cylindrical.

Meanwhile rolling performs well with the cylindrical objects but is more challenging for the lunch box. This is likely due to a failure to flip the lunch bag when it is laying flat on the table.

Spinning performs well on the two cylinder-like objects: the pill bottle and the tape roll. Meanwhile, spinning performs poorly on the frustum-like objects: the planters and the mug. One explanation is that when a cylinder is on its side it is fairly easy to rotate by pushing orthogonal to its rolling direction. Meanwhile, the same kind of push is likely to lead to rolling for a frustum.

V. DISCUSSION

The results show that the point-cloud based HACMan framework can be used with general reward functions instead of a specific goal position in order to train specific behaviors. The varying performance of the trained actions on different objects suggests that certain objects and likely certain object orientations afford different actions.

One extension to this work would be to train a neural network to predict the success rate of a given action with a given object point cloud. This would present the knowledge gained by the models trained in this paper in a convenient way that could be used by a high-level planner. The planner could query different paths through the different orientations of an object to reach a goal position. The planner would avoid taking difficult actions, like trying to flip up a face-down lunch box, and it would focus on easy actions, like rolling a pill bottle.

With some modifications, the action models could also be used to directly run a planner. The spin action could have an additional input for the size of the spin. The roll action could have an input for which face of the object should point up, and the push straight action could take in a vector telling the robot where to push the object. The planner could call the actions it desires as necessary. This would be different from using a planner with the original version of HACMan because HACMan just takes in a goal position, and there is no control over what actions are taken to get there.

VI. Limitations

Although this work shows some promising results, it has several limitations.

One major limitation of this work is that the trained models cannot be given a goal position or direction; they just try to change the object's position or orientation to maximize their respective reward functions. Future work could remove this limitation by adding additional inputs to the models.

There are also several limitations due to the model being trained and tested only in simulation. One limitation is the sim2real gap. The real world has higher uncertainty in friction and object dynamics, which would lead to lower performance for all the models, and especially the push straight model.

During training, there were several cases where the models started to exploit broken simulation physics. Steps were taken to avoid these exploits where they were spotted, but it is possible that the models learned some exploits, further increasing the expected sim2real gap.

Another limitation that comes from the simulation training is that in simulation, the robot could always distinguish the object's pose, while in the real world, multiple poses could lead to the same point cloud. For example, no matter how you orient a sphere its point-cloud looks the same. This could make it more difficult for the roll and spin actions to deal with highly symmetrical objects in the real world.

VII. Conclusion

This paper presents three RL models that use point cloud data to perform the push straight, roll, and spin non-prehensile manipulation actions. These models were able to learn meaningful object properties and their performance on different objects seems to be a strong indicator of the affordances of an object in a given state. Future work could modify the models or use the models' understanding of object affordances as part of a high-level planner to perform non-prehensile manipulation with a greater ability to plan ahead.

References

[1] W. Zhou, B. Jiang, F. Yang, C. Paxton, and D. Held, "HACMan: Learning Hybrid Actor-Critic Maps for 6D Non-Prehensile Manipulation," arXiv.org, Nov. 05, 2023. https://arxiv.org/abs/2305.03942