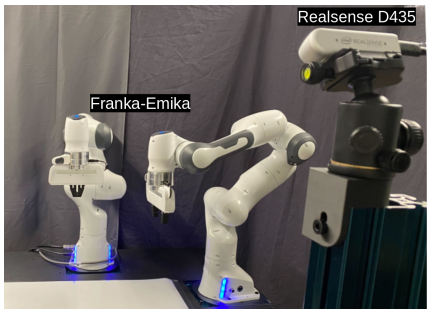


Figure 10: We used 2 Franka-emika robots and an Intel Realsense D435 for real-world setup.

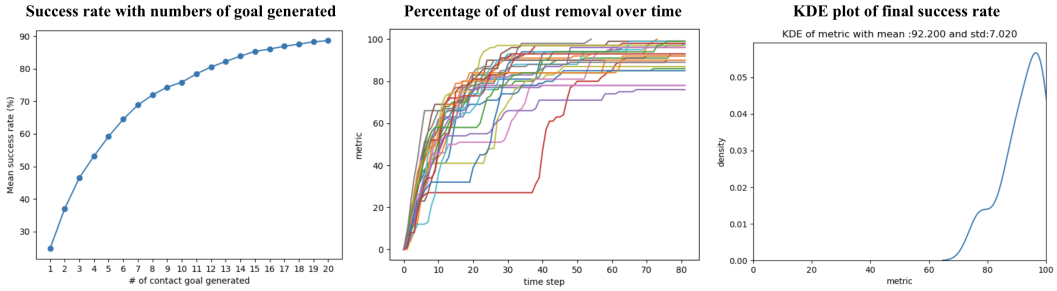


Figure 11: Details on wiping performance with training object over the number of goal generated (first) and time (second). Second plot visualizes each 25 tests' dust removal rate over time. Finally we show the distribution of success rate after 20 goals.

A Results Details

A.1 Realworld Setup Details

Fig. 10 shows our experimental setup and demonstrates the efficiency of our setup using a single accessible vision sensor from the front view. We note that only one arm was used for manipulation. We found that RGB image noise could adversely affect the planning performance of our method on the *sweep_to_dustpan* task. This is because the heatmap is sensitive to high frequency RGB noise, which is not present during simulation, making subsequent contact goal predictions on nearly identical inputs diverge. As such, for the *sweep_to_dustpan* task, and compliant variant, we repeated each trial twice and reported the better performing result. In the future, we hope to address this issue via data-augmentation on the heatmap scaling.

A.2 Real-world Wipe Desk

Fig. 12 shows generalization of our wiping task to unseen dot arrangements. This results in different heatmap distribution, As in the main experiment, we run the wiping until it generates 20 contact goals.

A.3 Push Button

Fig. 13 shows more example results on button pushing task with unseen numbers of buttons in the scene, unseen table elevation, and unseen prompt.

A.4 Compliant Sweep Task

Our method decouples contact goal generation from the low-level controller responsible for realizing the contact goals. To control contact between a deforming tool and the tabletop, we use the contact feature dynamics model from Van der Merwe et al. [21]. The model predicts contact geome-

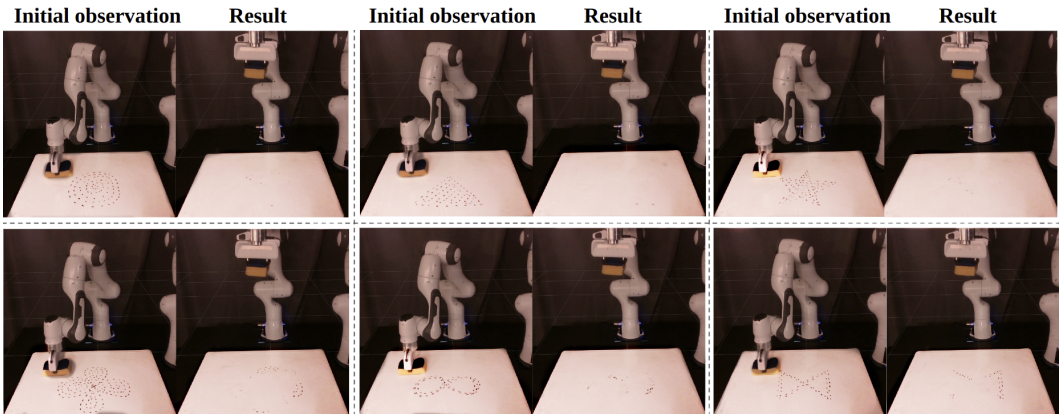


Figure 12: We examined the wiping results using the test1 object on different dot arrangements, leading to an unseen heatmap distribution for the model, which was originally trained only on square-shaped dot clusters. From the top left to the bottom right, we conducted tests on circle, triangle, star, quatrefoil, infinite, and hourglass shapes. Our observations show the first four shapes were nearly entirely erased after the wiping. In contrast, we observed a disparity in performance with the last two shapes, characterized by unfilled holes in their centers, leaving behind 10% and 54% of the dots, respectively.

tries (represented as lines in 3D) given candidate actions, conditioned on point cloud and wrench observations. The contact point cloud $\mathbf{P}_{t+i}^{dyn}$ is obtained by sampling evenly between the end points of the predicted contact line.

We train the model by performing randomized actions and label contact lines using a heuristic on the observed point clouds. Specifically, we threshold for points near the surface, then fit a line to the resulting points projected onto the tabletop. Point cloud observations are obtained by a Photoneo Phoxi 3D scanner (L) and a Photoneo MotionCam-3D Color (M+) scanner. Wrench observations are obtained by an ATI gamma force torque sensor, attached between the end effector and compliant tool. The dynamics model is trained on 3236 sampled transitions.

B Input Processing Details

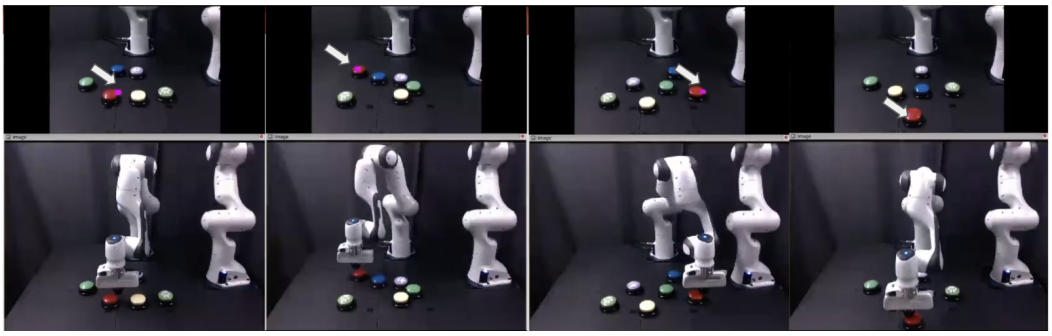
B.1 Heatmap Encoder

At the beginning of this project, we implemented our own version of 1D gray-scale image encoder based on pytorch's ResNet implementation by changing the first encoder's input dimension as 1 instead of 3. We used 4 residual blocks and 2 convolution layers for each residual blocks following the original pytorch implementation. Later in the project, we found that simply using a pretrained ResNet18 [27] trained on ImageNet gives faster convergence and lower training loss as shown in Fig. 14. The network is for RGB image, such that we repeated the grayscale heatmap inputs for three times and stacked in depth to match the desired the input dimension. We note that the pretrained image encoder's parameters are remained frozen without fining tuning.

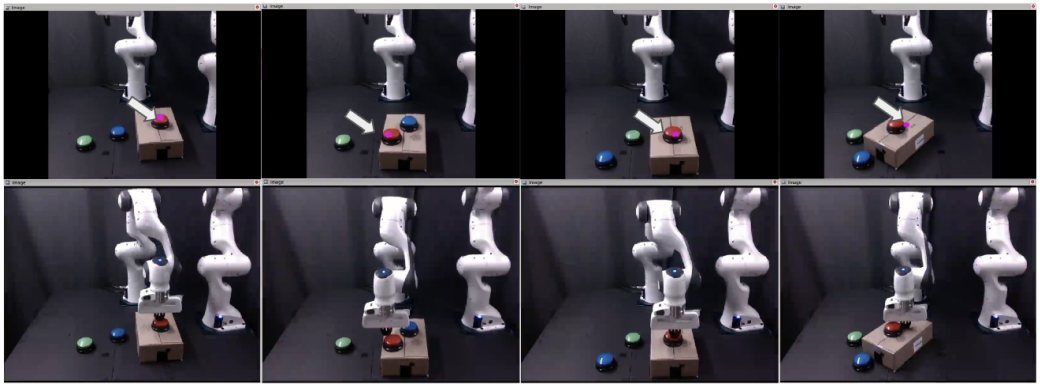
B.2 RGB Post-processing for Heatmap Generation

We masked the RGB input when generating the heatmap to display only the workspace, which is easily accessible with a camera calibrated to the robot. This is because we observed the presence of CLIP heatmap noises, particularly originating from verb words, and it is unrealistic to anticipate significant signals in regions beyond the robot's physical reach.

"Push the red button"



"Push the red button"



"Push the green button"

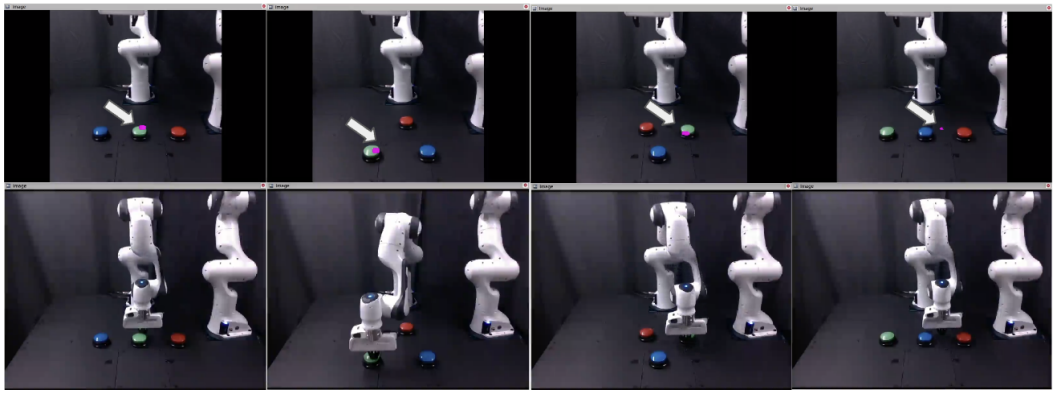


Figure 13: First row indicates generated contact goal indicated in magenta pixels. The second row indicates robot execution results.

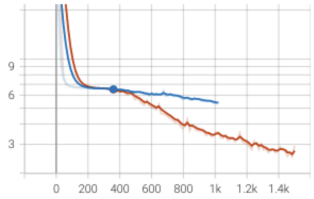


Figure 14: Blue is a training loss curve when training image encoder from scratch and the red is the trainind loss curve when training with frozen pretrained ResNet18. Other than the encoder, we used the same training settings. The dataset used for this experiment is 50 demonstrations of *press_button* dataset.

B.3 Data augmentation

During training, we applied 40 random heatmap augmentation of size 256×256 using flipping and translation(horizontal/vertical) ranging from 10 to 30 pixels.

C MPPI Details

C.1 Environment Geometry

We define environment as a pointcloud or the depthmap without tools and non-collidable objects. For simulation, we set *set_model_renderable(False)* at the task initialization, where *grasp_target* is the graspable objects. In real world, we set aside the tools from the camera angle at the task initialization and take a depth map to get the environment geometry.

C.2 Contact Patch Prediction

To obtain the intersections between the object point cloud and the environment, we followed these steps assuming the tool is close to the surface. Firstly, we transformed the transformed object point cloud to the world coordinates based on the given action. Next, we extracted the bottom 10% of the point cloud, which served as the contact candidates. These contact candidates were then projected onto the camera frame. Finally, we compared the world z-coordinate of each pixel belonging to the contact candidate with the z-coordinate of the environment. If the z-coordinate of the contact candidate was lower than or equal to that of the environment, we considered the candidate pixel to be in contact.

C.3 Intersection over Union

Our findings indicate that when using IoU metrics without offsetting the center, it can lead to undesirable yaw actions of the object. This occurs because the MPPI algorithm selects the action set with the highest yaw while attempting to reach the contact patch goal, even when a straight motion is required. This issue is particularly pronounced when the contact goal is far from the current object position, resulting in partial overlap between the contact goal and the MPPI prediction. To mitigate this problem, adjustments such as offsetting the center was necessary to ensure aligned tool movements.

C.4 MPPI Algorithm

Algorithm 1

$t \leftarrow 0$
$complete \leftarrow False$
while not $complete$ **do**
  if $cost \leq \delta$ **then** $\mathbf{C}_t = generate\_contact\_goal(obs)$
  end if
  $a_t \leftarrow mpc(s_t, \boldsymbol{C}_t)$
  $env.step(a_t)$ ▷ Move the agent
  $t \leftarrow t + 1$
  if $task.get\_completed$ **then**
    $complete \leftarrow True$ ▷ Success
  end if
  if $t < 100$ **then**
    $complete \leftarrow True$ ▷ Fail
  end if
end while

C.5 MPPI Parameter

MPPI parameter for simulation are the following: action_high =$[x, y, z, r, p, y]$=[0.04, 0.04, 0.002, 0.,0., 0.2], action_low=$[x, y, z, r, p, y]$=[-0.04, -0.04, -0.002, -0.,-0., -0.2], num_samples=600, nx=6, lambda=0.001, horizon=1, and $\sigma = \begin{bmatrix} 0.01 & 0. & 0. & 0. & 0. & 0. \\ 0. & 0.01 & 0. & 0. & 0. & 0. \\ 0. & 0. & 0.0005 & 0. & 0. & 0. \\ 0. & 0. & 0. & 0.0005 & & 0. \\ 0. & 0. & 0. & 0. & 0.0005 & 0. \\ 0. & 0. & 0. & 0. & 0. & 0.01 \end{bmatrix}$.