SAM-RL: Sensing-Aware Model-Based Reinforcement Learning via Differentiable Physics-Based Simulation and Rendering

Jun Lv[1], Yunhai Feng[2], Cheng Zhang[3], Shuang Zhao[3], Lin Shao[4*] and Cewu Lu[5*]

***Abstract*—Model-based reinforcement learning (MBRL) is recognized with the potential to be significantly more sample efficient than model-free RL. How an accurate *model* can be developed automatically and efficiently from raw sensory inputs (such as images), especially for complex environments and tasks, is a challenging problem that hinders the broad application of MBRL in the real world. In this work, we propose a sensing-aware model-based reinforcement learning system called *SAM-RL*. Leveraging the differentiable physics-based simulation and rendering, *SAM-RL* automatically updates the model by comparing rendered images with real raw images and produces the policy efficiently. With the sensing-aware learning pipeline, *SAM-RL* allows a robot to select an informative viewpoint to monitor the task process. We apply our framework to real world experiments for accomplishing three manipulation tasks: robotic assembly, tool manipulation, and deformable object manipulation. We demonstrate the effectiveness of *SAM-RL* via extensive experiments. Videos are available on our project webpage at https://sites.google.com/view/rss-sam-rl.**

I. Introduction

Over the past decade, deep reinforcement learning (RL) has resulted in impressive successes, including mastering Atari games [1], winning the games of Go [2], and solving Rubik's cube with a human-like robot hand [3]. However, deep RL algorithms adopt the paradigm of model-free RL and require vast amounts of training data, significantly limiting their practicality for real-world robotic tasks. Model-based reinforcement learning (MBRL) is recognized with the potential to be significantly more sample efficient than model-free RL [4].

How to automatically and efficiently develop an accurate *model* from raw sensory inputs, especially for complex environments and tasks, is a challenging problem that hinders the wide application of MBRL in the physical world.

One line of works [5, 6, 7, 8] adopt representation learning approaches to learn the *model* from raw input data. They aim to learn low-dimensional latent states and action representations from high-dimensional input data like images. But the learned deep network might not satisfy the physical dynamics, and its quality may also significantly degenerate beyond the training data distribution when testing in the wild. Recent developments in differentiable physics-based simulation [9, 10, 11, 12, 13, 14, 15, 16] and rendering [17, 18, 19, 20] provide an alternative direction to model the environment [21, 22]. Lv et al. [23] use differentiable physics-based simulation as the backbone of the *model* and train robots to perform articulated object manipulation in the real world. Their pipeline produces a file of Unified Robot Description Format (URDF) [24] of the environment, which is loaded into the differentiable simulation from raw point clouds gathered by an RGB-D camera mounted on its wrist. However, a sequence of camera poses is needed to scan the 3D environment every time step, and these camera poses are manually predefined, which is time-consuming and

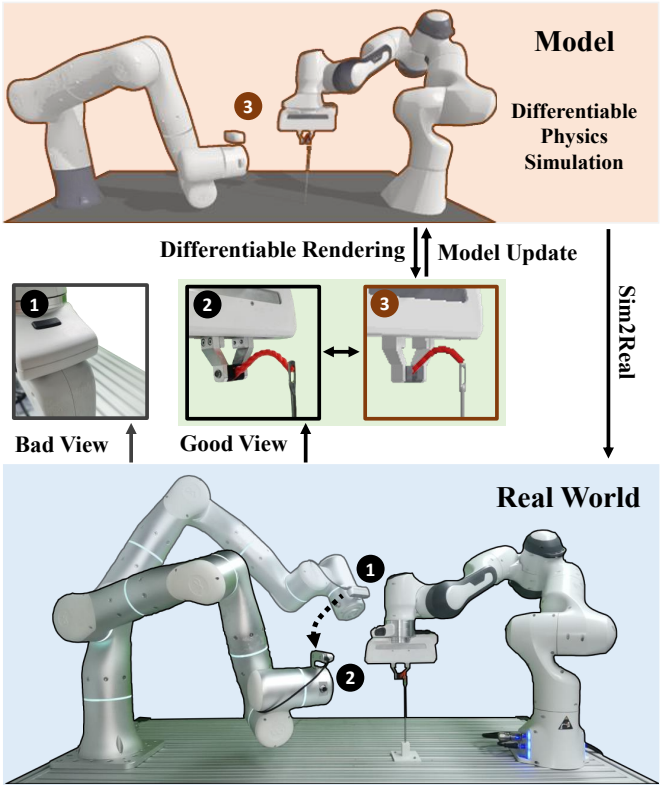


Fig. 1. Our proposed *SAM-RL* enables robots to autonomously select an informative camera viewpoint to better monitor the manipulation task (for example, the *Needle-Threading* task). We leverage the differentiable rendering to update the *model* by comparing the raw observation between simulation and the real world, and differentiable physics simulation to produce policy efficiently.

*Equal advising.
[1]Jun Lv is with the Department of Electronic Engineering, Shanghai Jiao Tong University, China. *[lyujune_sjtu@sjtu.edu.cn]*
[2]Yunhai Feng is with the Department of Computer Science and Engineering, University of California San Diego, USA. *[yuf020@ucsd.edu]*
[3]Cheng Zhang and Shuang Zhao are with the Department of Computer Science, University of California Irvine, USA. *[chengz20@uci.edu, shz@ics.uci.edu]*
[4]Lin Shao is with the Department of Computer Science, National University of Singapore, Singapore. *[linshao@nus.edu.sg]*
[5]Cewu Lu is the corresponding author, the member of Qing Yuan Research Institute and MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University, China and Shanghai Qi Zhi Institute. *[lucewu@sjtu.edu.cn]*

difficult to adapt to various tasks. Object colors and geometric details are not included in the *model*, limiting its representation capability [23].

By integrating differentiable physics-based simulation and rendering, we propose a sensing-aware model-based reinforcement learning system called *SAM-RL*. As shown in Fig. 1, we apply *SAM-RL* on a robot system with two 7-DoF robotic arms (Flexiv Rizon [25] and Franka Emika Panda), where the former mounts an RGB-D camera, and the latter handles manipulation tasks. Our framework is sensing-aware, which allows the robot to automatically select an informative camera view to effectively monitor the manipulation process, providing the following benefits. First, the system no longer requires obtaining a sequence of camera poses at each step, which is extremely time-consuming. Second, compared with using a fixed view, *SAM-RL* leverages varying camera views with potentially fewer occlusions and offers better estimations of environment states and object status (especially for deformable bodies). The improved quality in object status estimation contributes more effective robotic actions to complete various tasks. Third, by comparing rendered and measured (i.e., real-world) images, discrepancies between the simulation and the reality are better revealed and then reduced automatically using gradient-based optimization and differentiable rendering.

In practice, we train the robot to learn three challenging manipulation skills: *Peg-Insertion*, *Spatula-Flipping*, and *Needle-Threading*. Our experiments indicate that *SAM-RL* can significantly reduce training time and improve success rate by large margins compared to common model-free and model-based deep reinforcement learning algorithms.

Our primary contributions include:

- proposing an active-sensing framework named *SAM-RL* that enables robots to select informative views for various manipulation tasks;
- introducing a model-based reinforcement learning algorithm to produce efficient policies;
- conducting extensive quantitative and qualitative evaluations to demonstrate the effectiveness of our approach;
- applying our framework to robotic assembly, tool manipulation, and deformable object manipulation tasks both in simulation and real world experiments.

II. Related Work

We review related literature on key components in our approach, including model-based reinforcement learning, next best view, integration of differentiable physics-based simulation and rendering, and robotic manipulation. We describe how we are different from previous work.

A. Model-based Reinforcement Learning

MBRL is considered to be potentially more sample efficient than model-free RL [4]. However, automatically and efficiently developing an accurate *model* from raw sensory data is a challenging problem, which retards MBRL from being widely applied in the real world. For a broader review of the field on MBRL, we refer to [26]. One line of works [5, 6, 7, 8] use representation learning methods to learn low-dimensional latent state and action representations from high-dimensional input data. But the learned models might not satisfy the physical dynamics, and the quality may also significantly drop beyond the training data distribution. Recently, Lv et al. [23] leveraged the differentiable physics simulation and developed a system to produce a URDF file to model the surrounding environment based on an RGB-D camera. However, the RGB-D camera poses used in [23] are predefined and can not adjust to different tasks. Our approach allows the robot to select the most informative camera view to monitor the manipulation process and update the environment *model* automatically.

B. Next Best View in Active Sensing

Next Best View (NBV) has been one of the core problems in active sensing. It studies the problem of how to obtain a series of sensor poses to increase the information gain. The information gain is explicitly defined to reflect the improved perception for 3D reconstruction [27, 28, 29, 30], object recognition [31, 32, 33, 34], 3D model completion [35], and 3D exploration [36, 37]. Unlike perception-related tasks, we explore the NBV over a wide range of robotic manipulation tasks. Information gain in the robotic manipulation tasks is difficult to define explicitly and is implicitly related to task performance. In our system, the environment changes accordingly after the robot's interaction. We integrate the information gain into the Q function to reflect the informative viewpoint for manipulation.

C. Integration of Differentiable Physics-Based Simulation and Rendering

Recently, great progresses have been made in the field of differentiable physics-based simulation and rendering. For a broader review, please refer to [9, 10, 11, 12, 13, 14, 15, 16] and [38, 39]. With the development of these techniques, Jatavallabhula et al. [21] first proposed a pipeline to leverage differentiable simulation and rendering for system identification and visuomotor control. Ma et al. [22] introduced a rendering-invariant state predictor network that maps images into states that are agnostic to rendering parameters. By comparing the state predictions obtained using rendered and ground-truth images, the pipeline can backpropagate the gradient to update system parameters and actions. Sundaresan et al. [40] proposed a real-to-sim parameter estimation approach from point clouds for deformable objects. Different from these works, we use the differentiable simulation and rendering to find the next best view for various manipulation tasks and update the object status in the *model* by comparing rendered and captured images.

D. Manipulation

Our framework can be adopted to improve the performance of a range of manipulation tasks. We review the related work in these domains. *1) Peg-insertion.* Peg insertion is a classic robotic assembly task with rich literature [41, 42, 43]. For a

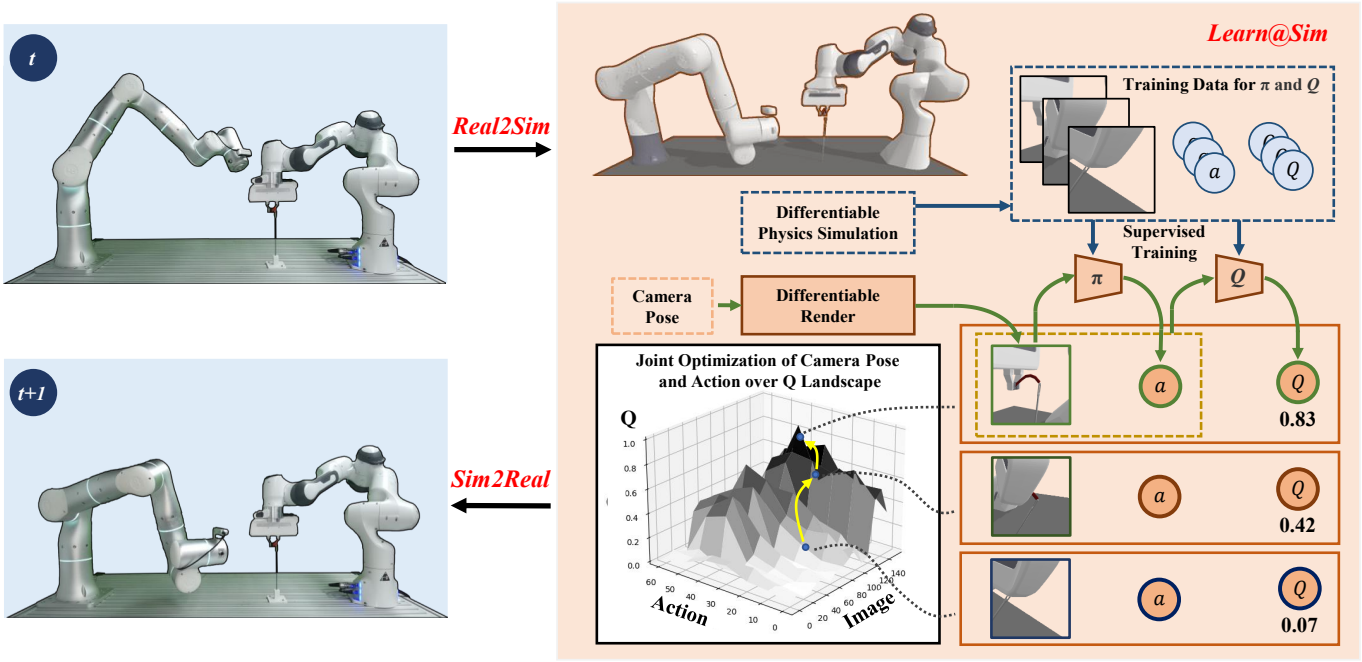


Fig. 2. The overall approach of *SAM-RL* includes Real2Sim, Learn@Sim, and Sim2Real stages. *SAM-RL* automatically develop and update the *model* during Real2Sim stage. During the Learn@Sim stage, it learns the sensing-aware Q function and actor π^{sim} in the *model*. The differentiable physics simulation generates training data (rendered image, action, and associated return) to learn the Q function and actor function, which allows the robot to select an informative view. In the Sim2Real stage, *SAM-RL* learns a residual policy to reduce the sim-to-real gap.

broad review of peg-insertion, we refer to [44]. *2) Spatula-Flipping.* Chebotar et al. [45] used the tactile sensor to train the robot learning to perform a scraping task with a spatula. Tsuji et al. [46] studied the dynamic object manipulation by a spatula. They clarified the conditions for achieving dynamic movements and presented a unified algorithm for generating a variety of movements. *3) Needle-Threading.* The needle threading task requires the robots to adapt action according to the thread deformation. Silvério et al. [47] relied on a high-resolution laser scanner to perceive the thread and needle. Huang et al. [48] used a high-speed camera to monitor the process and provide high-speed visual feedback. Kim et al. [49] proposed a deep imitation learning algorithm for the needle threading task. Unlike approaches above, we develop a sensing-aware model-based reinforcement learning approach to learn these skills.

III. Technical Approach

Given a manipulation task denoted as $\mathcal{T}$, our pipeline takes as input images gathered from an RGB-D camera and outputs a policy to select the camera pose $\mathcal{P}^c$ followed by producing an action a. An overview of our proposed method is shown in Fig. 2. In what follows, we first briefly introduce the *model* $\mathcal{M}$ that integrates differentiable physics-based simulation and rendering in Sec. III-A. Then, we describe developing and updating the *model* in $\mathcal{M}$ (Real2Sim) in Sec. III-B, training robots to learn the perception and action with the *model* (Learn@Sim) in Sec. III-D, and applying the learned *model* to the real world (Sim2Real) in Sec. III-C.

A. Model with Differentiable Simulation and Rendering

In this work, we combine the differentiable physics-based simulation and rendering. The resulting differentiable system plays the backbone of the *model*, which we denote as $\mathcal{M}$, for the model-based reinforcement learning. The *model* can load robots, cameras, and objects denoted as $\{\mathcal{O}_j^{sim}\}$ along with their visual/geometric (e.g., shape, pose, and texture) and physical attributes (e.g., mass and inertial). We denote the attributes of all objects loaded in the simulation as one type of *model*'s parameters $\psi_{\mathcal{M}}$ as follows:

$$\psi_{\mathcal{M}} = \sum_j \mathcal{O}_j^{sim}. \tag{1}$$

The *model* can render an image $\mathcal{I}^{sim}$ under the camera pose $\mathcal{P}^c$ and *model* parameter $\psi_{\mathcal{M}}$ through:

$$\mathcal{I}^{sim} = \mathcal{M}(\psi_{\mathcal{M}}, \mathcal{P}^c; \textit{render}) \tag{2}$$

We can get the gradients $\partial\mathcal{I}^{sim}/\partial\mathcal{P}^c$ and $\partial\mathcal{I}^{sim}/\partial\mathcal{O}_j^{sim}$ using differentiable rendering [18]. Note that $\partial\mathcal{I}^{sim}/\partial\mathcal{O}_j^{sim}$ contains only the gradient with respect to object visual and geometric attributes.

Additionally, given the state s_t^{sim} including the object attributes $\psi_{\mathcal{M}}$ and the robots' status, when an action denoted as a_t^{sim} is executed (for example, an external force is exerted on the object), the *model* simulates the next state via

$$s_{t+1}^{sim} = \mathcal{M}(s_t^{sim}, a_t^{sim}; \textit{forward}) \tag{3}$$

in a differentiable fashion [10], providing the gradients $\partial s_{t+1}^{sim}/\partial a_t^{sim}$ and $\partial s_{t+1}^{sim}/\partial s_t^{sim}$.

B. Real2Sim: Developing Model from the Real World

1) Build the Initial Model: For model-based reinforcement learning, the first step is to build an initial *model* of the environment. The initial *model* does not need to be accurate and can be created using current 3D object reconstruction methods with RGB-D cameras like BundleFusion [50] and KinectFuction [51]. In our setting, as shown in Fig. 2, a calibrated RGB-D camera is mounted on the robot's wrist. Therefore the robot system takes a set of images with corresponding accurate camera poses. The initial *model* can also be built directly from a CAD model [52] or following the pipeline described in SAGCI [23] to produce the URDF. After the initialization, the *model* $\mathcal{M}$ contains the robots, objects and an RGB-D camera.

2) Update the Model with Differentiable Siumulation and Rendering: After having an initial *model*, we then describe how to update the *model* by directly comparing the raw visual observation in simulation and the real world, leveraging the differentiable simulation [10] and rendering [18]. In this work, we only care about one object and assume we can get accurate object segmentation of real world images. With common techniques such as Mask R-CNN [53], it's feasible to get a fine object-level instance segmentation.

At the beginning, we update the camera and robot pose in simulation to match the corresponding camera and robot pose in the real world. Then we get the rendered RGB-D image from $\mathcal{M}$ and corresponding real RGB-D image with associated segmentation. Based on the camera parameters, we transform the depth image to point cloud and segment the point cloud. We denote the RGB image, associated object segmentation, and segmented point cloud ($\mathcal{I}^{sim,rgb}$, $\mathcal{G}^{sim}$, $\mathcal{X}^{sim}$) in the simulation and ($\mathcal{I}^{real,rgb}$, $\mathcal{G}^{real}$, $\mathcal{X}^{real}$) in the real world. For simplicity, we use $\mathcal{I}^{sim}$ to represent the ($\mathcal{I}^{sim,rgb}$, $\mathcal{G}^{sim}$, $\mathcal{X}^{sim}$) and $\mathcal{I}^{real}$ is defined accordingly.

We define the loss functions as follows.

$$\mathcal{L}_1 = \|\mathcal{G}^{real} \otimes \mathcal{I}^{real,rgb} - \mathcal{G}^{sim} \otimes \mathcal{I}^{sim,rgb}\|_1 \tag{4}$$

$$\mathcal{L}_2 = \textit{EMD}(\mathcal{X}^{sim}, \mathcal{X}^{real}) \tag{5}$$

$$\mathcal{L} = \lambda_1 \mathcal{L}_1 + \lambda_2 \mathcal{L}_2 \tag{6}$$

where $\otimes$ represents the pixel-wise product operator and *EMD* represents the Earth Mover Distance [54] to measure the distance between two 3D point clouds. Note that $(\mathcal{I}^{sim,rgb}, \mathcal{G}^{sim}, \mathcal{X}^{sim}) = \mathcal{M}(\psi_{\mathcal{M}}, \mathcal{P}^c; \textit{render})$, as explained in Sec. III-A, is differentiable. With the gradient $\partial\mathcal{L}/\partial\psi_{\mathcal{M}}$, we can update the *model* $\mathcal{M}$'s parameters $\psi_{\mathcal{M}}$ including object mesh vertices, colors, and poses so that the loss $\mathcal{L}$ is reduced.

$$\psi_{\mathcal{M}} \leftarrow \psi_{\mathcal{M}} - \lambda_{\mathcal{M}} \frac{\partial \mathcal{L}}{\partial \psi_{\mathcal{M}}} \tag{7}$$

During the manipulation, at each time step t, the *model* parameters like the poses would be changed by the robot action a_t. We denote the object poses as $\mathcal{P}_t^{\mathcal{O}}$. We will first update the object pose via the forward simulation by

$$\mathcal{P}_{t+1}^{\mathcal{O}} \leftarrow \mathcal{M}(\mathcal{P}_t^{\mathcal{O}}, a_t, forward) \tag{8}$$

However, the sim-to-real gap may make the *model* pose in simulation inaccurate. So we still need to use the method described in 7 to further update the *model* pose.

Next, we describe how to update objects' physical attributes. Starting from poses $\mathcal{P}_0^{\mathcal{O}}$ at timestep $t = 0$, with a sequence of actions $\{a_t\}_{t=0}^{T}$ applied, the simulation calculates the object poses at timestep $t = T$ denoted as $\mathcal{P}_T^{\mathcal{O}}$ based on objects' physical attributes by simulating Eqn. 8 for T steps.

$$\mathcal{P}_T^{\mathcal{O}} \leftarrow \mathcal{M}(\mathcal{P}_0^{\mathcal{O}}, \{a_t\}_{t=0}^{T}, forward) \tag{9}$$

The object poses from the real-world at timestep $t = T$ are gathered with differentiable rendering denoted as $\tilde{\mathcal{P}}_T^{\mathcal{O}}$. We then calculate the distance between the two poses.

$$\mathcal{L}_p = \|\tilde{\mathcal{P}}_T^{\mathcal{O}} - \mathcal{P}_T^{\mathcal{O}}\| \tag{10}$$

Then object physical attributes $\psi_{\mathcal{M}}^{phy}$ (mass and inertial) are updated to minimize the loss $\mathcal{L}_p$.

$$\psi_{\mathcal{M}}^{phy} \leftarrow \psi_{\mathcal{M}}^{phy} - \lambda_{\mathcal{M}} \frac{\partial \mathcal{L}_p}{\partial \psi_{\mathcal{M}}^{phy}} \tag{11}$$

Through this, we can keep reducing the discrepancy between the simulation and the real world, making the *model* more and more accurate. Up to now, we have introduced how to build and update the *model*.

C. Sim2Real: Learning Residual Policy in the Real World

We delay the discussion of how to learn the policy to complete the task $\mathcal{T}$ with the *model* in the simulation to the next subsection III-D. Here we assume that we have the policy π^{sim} in simulation which takes rendered images $\mathcal{I}^{sim}$ as inputs and outputs actions denoted as

$$a^{sim} = \pi^{sim}(\mathcal{I}^{sim}) \tag{12}$$

We describe how to apply the learned policy from simulation to the real world by learning a residual policy π^{res} to reduce the sim-to-real gap in this subsection.

We set the same camera pose denoted as $\mathcal{P}^c$ both in the simulation and the real world and get images denoted as $\mathcal{I}^{sim,0}$ and $\mathcal{I}^{real}$. We first update the *model*'s parameter $\psi_{\mathcal{M}}$ by minimizing the loss with Eqn. 6 as described in Sec. III-B2 and then get the new images $\mathcal{I}^{sim}$ with the updated $\psi_{\mathcal{M}}$.

With the $\mathcal{I}^{sim}$ and $\mathcal{I}^{real}$, we get an action from simulation a^{sim}. Instead of directly applying the action a^{sim} in the real world, we train a residual policy that takes in the real image $\mathcal{I}^{real}$ and the a^{sim}, outputs the residual action

$$\delta a^{real} = \pi^{res}(\mathcal{I}^{real}, a^{sim}) \tag{13}$$

Then actor model in the real world gets the action as follows.

$$a^{real} = \pi^{real}(\mathcal{I}^{real}, a^{sim}) = a^{sim} + \delta a^{real} \tag{14}$$

We follow the training process of residual policy described in [55]. The residual policy should never make a good initial policy worse. We therefore initialize the residual policy so that

$$\pi^{res}(\mathcal{I}^{real}, a^{sim}) = 0 \tag{15}$$

before training. We do this by initializing the last layer of the network to be zero. Once the real world action a^{real} is executed, we receive the next image $\mathcal{I}^{real'}$ and the reward

$$r^{real} = \begin{cases} 1 & succeed \\ 0 & otherwise \end{cases} \tag{16}$$

The task will be considered *done* when it succeeds or exceeds the max action step number. To train the residual policy π^{res}, we store the $(\mathcal{I}^{real}, a^{real}, a^{sim}, r^{real}, done, \mathcal{I}^{real'})$ to the reply buffer to update the π^{res} following a common deep reinforcement learning procedure *TD3* [56], an actor-critic deep RL method.

D. Learn@Sim: Learning Sensing-aware Action in Simulation

With the *model*, we discuss how to learn the informative camera pose $\mathcal{P}^c$ associated with the rendered image $\mathcal{I}^{sim}$ and action a^{sim} to complete the given task $\mathcal{T}$ in the simulation.

1) Sensing-aware Q-function: We adopt the Q function $Q^{sim}(\mathcal{I}^{sim}, a^{sim})$ to reflect the informative viewpoint. We can calculate the gradient of Q^{sim} with respect to the camera pose $\mathcal{P}^c$ as follows.

$$\frac{\partial Q^{sim}(\mathcal{I}^{sim}, a^{sim})}{\partial \mathcal{P}^c} = \frac{\partial Q^{sim}(\mathcal{I}^{sim}, a^{sim})}{\partial \mathcal{I}^{sim}} \frac{\partial I^{sim}}{\partial \mathcal{P}^c} \tag{17}$$

Here the first term $\partial Q^{sim}/\partial \mathcal{I}^{sim}$ is available through the backward propagation of the neural network. The second term $\partial I^{sim}/\partial \mathcal{P}^c$ can be obtained from the differentiable renderer [18]. The gradient $\partial Q^{sim}/\partial \mathcal{P}^c$ provides information on how to find a more informative viewpoint, which makes the pipeline sensing-aware. Under the more informative viewpoint, the actor has a better sense of the state to generate an action associated with a higher Q value. We verify our hypothesis by visualizing the learned Q function in Sec IV-B2.

2) Learning Actor in Simulation to Reflect the Sensing Quality in the Real World: In this part, we discuss how we learn an actor π^{sim} in simulation, which takes rendered image $\mathcal{I}^{sim}$ and outputs the action a^{sim}. There are multiple ways to learn the π^{sim} in the simulation via reinforcement learning or imitation learning. In this work, we choose the imitation learning approach to learn the π^{sim}, which we find effective and efficient. With the *model* $\mathcal{M}$ built with the differentiable physics simulation [10], our pipeline generates trajectories completing the tasks $\{(s_t^{sim}, a_t^{sim}|\mathcal{T})\}_{t=1}^{T}$ to train the actor in simulation π^{sim}. To effectively generate the trajectories inside the differentiable physics simulation [10], we follow the method mentioned in [23]. Given task $\mathcal{T}$, we optimize the trajectory via the following equations.

$$\mathcal{L}(\mathcal{T}) = \sum_{t=0}^{T-1} l(s_t^{sim}, a_t^{sim}; \mathcal{T}) \tag{18}$$

$$\text{s.t. } s_{t+1}^{sim} = \mathcal{M}(s_t^{sim}, a_t^{sim}, forward) \tag{19}$$

$$s_0^{sim} = s^{\text{init}} \tag{20}$$

Here we explain the loss function in Eqn. 18.

$$l(s_t^{sim}, a_t^{sim}; \mathcal{T}) \tag{21}$$

$$= \begin{cases} \|a_t^{sim}\|^2 & t < T_1 - 1 \\ \|s_{T_1-1}^{sim} - \mathcal{G}(\mathcal{T})_1\|^2 & t = T_1 - 1 \\ \vdots & \\ \|a_t^{sim}\|^2 & T_{n-1} - 1 < t < T_n - 1 \\ \|s_{T_n-1}^{sim} - \mathcal{G}(\mathcal{T})_n\|^2 & t = T_n - 1 \end{cases} \tag{22}$$

$\mathcal{G}(\mathcal{T})$ is the goal of the task. to make it easier to generate the trajectories, we define n sub-goals for each task. In practice, instead of directly optimizing the whole trajectory, the pipeline will optimize the trajectory to achieve each sub-goal in sequence. Once a goal is achieved, the pipeline will move on to the next goal.

After gathering these trajectories $\{(s_t^{sim}, a_t^{sim}|\mathcal{T})\}_{t=1}^{T}$ in the simulation, we rendered multiple images $\{\mathcal{I}_t^{sim}(\mathcal{P}^{c,i})\}$ for each state s_t^{sim} under different camera poses $\{\mathcal{P}^{c,i}\}$. We train an actor $\pi^{sim}(\mathcal{I}_t(s_t^{sim}, \mathcal{P}^{c,i}))$ to predict the action $\hat{a}_t^{sim}$ to imitate the corresponding action a_t^{sim}. In the process, the prediction error is defined as follows

$$\mathcal{L}^{\pi^{sim}} = \|\hat{a}_t^{sim} - a_t^{sim}\|_2 \tag{23}$$

During the learning process, the prediction error also depends on the sensing quality. Our hypothesis is the actor might not learn the effective ground-truth action if there is insufficient state information in the rendered image. For example, if the camera is always looking into the sky and the object does not appear in the rendered image, it will be difficult for the actor to learn the correct action. We visualize the prediction error in Sec.IV-B2. Note that the learning process also generates success and failure trajectories, which are used to learn the Q function.

3) Learning Sensing-aware Q Function: Here we describe how to learn the sensing-aware Q function denoted as $Q^{sim}(\mathcal{I}^{sim}, a^{sim})$. There are also multiple ways to learn the Q function. In this work, we formulate it as a supervised learning problem. We roll out the trajectories in the simulation to generate the training data. Given a trajectory $\{(\mathcal{I}_t^{sim}(\mathcal{P}_t^{c,i}), \hat{a}_t^{sim})\}_{t=1}^{T}$, we calculate the return $\mathcal{R}_t = \sum_{i=t}^{T} \gamma^{T-i} r_i$ associated with each image and action pair, where r_t is the reward with same definition as Eqn. 16 and γ is the discount factor. We then train a deep network that takes $\mathcal{I}_t^{sim}$ and $\hat{a}_t^{sim}$ as inputs and outputs the Q value by minimizing the following loss.

$$\mathcal{L}^{Q} = \|Q^{sim}(\mathcal{I}_t^{sim}, \hat{a}_t^{sim}) - \mathcal{R}_t\|_2 \tag{24}$$

4) Selecting Perception and Action Leveraging the Actor and Q-function: Starting from an initial viewpoint $\mathcal{P}^c$, the simulation rendered an image $\mathcal{I}^{sim}$. We feed the image into the actor model to get an action $a^{sim} = \pi^{sim}(\mathcal{I}^{sim})$, and then send the image and action to the Q function to get the value $Q^{sim}(\mathcal{I}^{sim}, a^{sim})$. We get the gradient with respect to the $\mathcal{P}^c$ and update the camera pose as follows.

$$\mathcal{P}^{c'} = \mathcal{P}^c + \lambda \frac{\partial Q^{sim}(I^{sim}, a^{sim})}{\partial \mathcal{P}^c} \tag{25}$$

Algorithm 1: Overall algorithm

Input: the model $\mathcal{M}$ with its model parameters $\psi_{\mathcal{M}}$, camera pose $\mathcal{P}$, take the real image using real camera $Cam(\mathcal{P})$

1 **for** *t in each iteration* **do**
2 $\quad \mathcal{I}^{sim} \leftarrow \mathcal{M}(\psi_{\mathcal{M}}, \mathcal{P}; render)$, $\mathcal{I}^{real} \leftarrow Cam(\mathcal{P})$;
3 $\quad$ Update $\psi_{\mathcal{M}}$ by reducing $\|\mathcal{I}^{sim} - \mathcal{I}^{real}\|$;
4 $\quad \mathcal{I}^{sim} \leftarrow \mathcal{M}(\psi_{\mathcal{M}}, \mathcal{P}; render)$, $a^{sim} = \pi^{sim}(\mathcal{I}^{sim})$;
5 $\quad$ **while** *True* **do**
6 $\quad\quad \mathcal{P}' = \mathcal{P} + \lambda_{\mathcal{P}} \frac{\partial Q(I^{sim}, a^{sim})}{\partial \mathcal{P}}$;
7 $\quad\quad \mathcal{I}^{sim'} \leftarrow \mathcal{M}(\psi_{\mathcal{M}}, \mathcal{P}'; render)$, $\mathcal{I}^{real'} \leftarrow Cam(\mathcal{P}')$;
8 $\quad\quad$ Update $\psi_{\mathcal{M}}$ by reducing $\|\mathcal{I}^{sim'} - \mathcal{I}^{real'}\|$;
9 $\quad\quad \mathcal{I}^{sim'} \leftarrow \mathcal{M}(\psi_{\mathcal{M}}, \mathcal{P}'; render)$, $a^{sim'} = \pi^{sim}(\mathcal{I}^{sim'})$;
10 $\quad\quad$ **if** $Q^{sim}(\mathcal{I}^{sim}, a^{sim}) \leq Q^{sim}(\mathcal{I}^{sim'}, a^{sim'})$ **then**
11 $\quad\quad\quad \mathcal{I}^{real} \leftarrow \mathcal{I}^{real'}$, $a^{sim} \leftarrow a^{sim'}$;
12 $\quad\quad\quad$ break;
13 $\quad\quad$ **end**
14 $\quad\quad \mathcal{I}^{sim} \leftarrow \mathcal{I}^{sim'}, \mathcal{P} \leftarrow \mathcal{P}'$, $a^{sim} \leftarrow a^{sim'}$;
15 $\quad$ **end**
16 $\quad$ **Add the updated Model with $\psi_{\mathcal{M}}$ to data buffer to generate trajectories**;
17 $\quad$ **Update the actor and Q function**;
18 $\quad a^{real} = \pi^{real}(\mathcal{I}^{real}, a^{sim})$;
19 $\quad$ Execute the action a^{real} both in the real world and in simulation;
20 $\quad$ **Add the transition data into the replay buffer to sim2real**;
21 $\quad$ **Update the residual policy**
22 **end**

With the new camera pose $\mathcal{P}^{c'}$, we gather the new rendered image $\mathcal{I}^{sim'}$, action $a^{sim'} = \pi^{sim}(\mathcal{I}^{sim'})$, and the associated $Q^{sim}(\mathcal{I}^{sim'}, a^{sim'})$. We accept the new camera pose $\mathcal{P}^{c'}$ and new action $a^{sim'}$ if $Q(\mathcal{I}^{sim'}, a^{sim'}) \geq Q(\mathcal{I}^{sim}, a^{sim})$.

E. Overall Learning Process

We summarize the overall test stage pipeline in the following Alg. 1. Given an initial *model*, we leverage the differentiable rendering and simulation to update the *model* parameter $\psi_{\mathcal{M}}$ and select the camera pose (Line 1-15).

Our algorithm add the update *model* $\mathcal{M}$ into our data buffer. Then our framework starts to generate the trajectories to accomplish the according task starting from the state described by the *model* $\mathcal{M}$ associated with the parameter $\mathcal{M}$. (Line 16)

We calculate the return value and expert actions to train the actor network and Q network in simulation. (Line 17)

Then we train the residual policy to deal with the sim2real gap. (Line 18-21)

IV. Experiments

In this section, we introduce our experimental setup and conduct quantitative and qualitative evaluations to demonstrate the effectiveness of our approach. Our experiments focus on evaluating the following questions

- Can our proposed selecting perception and action procedure described at III-D4 improve the performance of various manipulation tasks?
- How does our *SAM-RL* approach compare with model-free and model-based deep reinforcement learning algorithms?
- How effective is each component in our proposed *SAM-RL* algorithm?
- How effective is the *model* update process leveraging the differentiable simulation and rendering?

A. Experimental Setup

1) Real World: We set up the real world experiment with two 7 DoF robotic arms as shown in Fig 1. One robot is the Franka Panda performing the manipulation task. The other robot is Flexiv Rizon [25] holding the RGB-D RealSense camera. We calibrate the camera intrinsic matrix and the hand-eye transformation between the camera and the Flexiv's end-effector. We also measure the relative transformation between two robots' bases in the world coordinate.

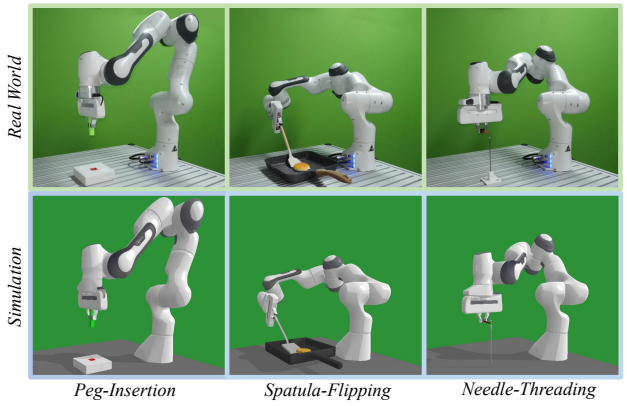


Fig. 3. Visualization of experiment setup for *Peg-Insertion*, *Spatula-Flipping*, and *Needle-Threading*.

2) Simulation: In the simulation, we set the camera intrinsic matrix and relative transformations of the camera to Flexiv and Flexiv to Franka based on the real-world calibration results. We use PyBullet [57] to simulate the real world, which is different from our differentiable physics simulation for quantitative experiments. Objects are initialized with a random pose within manually defined bounds as described following.

3) Differentiable Physics Simulation and Rendering: We combine the differentiable physics-based simulation and rendering to model the real world, update objects attributes, calculate expert trajectories, and optimize camera pose. We use NimblePhysics [10] as differentiable simulation and Redner [18] as differentiable renderer.

4) Manipulation Tasks: We design three different manipulation tasks as shown in Fig. 3, and the goal of each task as shown in Fig. 4.

Peg-Insertion, which is inserting a peg into a hole. We assume the robot holds the peg tightly, so the state of the task is the translation of the peg, which is 3-dimension and could be calculated via robot forward kinematic. And the hole is initialized at a random location by $10cm \times 10cm$ both in simulation and real world. The task is considered successful if the peg is inserted into the hole. Our pipeline adopt an automatic termination function in real world by comparing

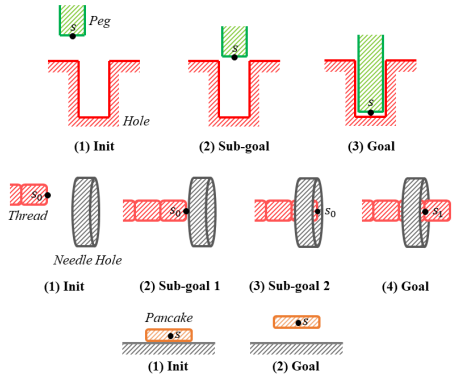


Fig. 4. The goal of each task. The first row is *Peg-Insertion*, the second is *Needle-Threading*, and the third is *Spatula-Flipping*.

the distance between the current state and goal state of the peg. The max number of action steps is 100.

Spatula-Flipping, which is flipping a pancake with a spatula. The state of the task is the translation of the pancake, which is 3-dimension. The pancake is initialized at a random location by $2cm \times 2cm$ both in simulation and real world. The *Spatula-Flipping* is successful if the pancake is lifted up by the spatula and then flipped. We track the position of the pancake via the method mentioned in Sec. III-B2. Our pipeline also adopt an automatic termination function for *Spatula-Flipping* in real world by comparing the distance between the current state and goal state. The max number of action steps is 200.

Needle-Threading, which is threading a needle. In simulation, the thread is simulated using 10 links. There are two revolute joints (See Fig. 5) between the links next to each other. The state of the task is the position of each link of the thread, which is 30-dimension in total. In the real world, the thread is manually randomly bent to initialize. While in simulation, for each revolute joint described in Fig. 5, the state of the joint is initialized ranging from $-10°$ to $10°$. We manually decide success for *Needle-Threading*, because it is hard to detect whether the thread is through the needle hole automatically with high precision. The max number of action steps is 100.

Fig. 5. There are two revolute joints between links, rotate along the x and y axis (blue and red).

5) Training Details: The size of the input images is $128 \times 128 \times 6$ (*RGB-XYZ*), while the size of the action space is 3, we only enable the translation of the Franka's end-effector and disable the rotation. For both the actor and critic (Q function) of π^{sim} and π^{res}, the network contains a CNN feature extractor and a MLP head. The feature extractor has 5 layers to extract a 128-dimensional vector from images, while the MLP head has 3 layers takes in the extracted vector, outputs the action or Q value. Note that other inputs, like the predicted action for the critic network or the base action for residual actor, will be concatenated with the images and input to the feature extractor.

In real world experiments, we train the residual policy network for 100 episodes for *Needle-Threading* and 10 episodes for *Peg-Insertion* and *Spatula-Flipping*. To better reduce the sim-to-real gap, we also augment the training image $\mathcal{I}^{sim}$ by adding noise to the *RGB* and *XYZ* value of each pixel to imitate the sensor noise.

B. Evaluating the Sensing-aware Function

1) Quantitative Results: To evaluate the learned sensing-aware Q function during Learn@Sim stage, we set up the following experiments inside the differentiable physics simulation. We use the same actor model π^{sim} trained with rendered images under multiple camera views, and evaluate the actor with and without leveraging the sensing-aware Q function to optimize the camera pose during manipulation processes, denoted as *Ours* and *Ours(w/o pose-opt)*, respectively. We also train the same actor with rendered images under a fixed camera view, and evaluate the trained actor under the same camera view, denoted as *Fixed-View*. To make relatively fair compassion, we train the actor in *Fixed-View* to have the same prediction error in the training process as in *Ours*. We report the task success rate based on 100 experiments under these three settings. As shown in Tab. I, our pipeline can find the informative view to benefit robot manipulation with the sensing-aware Q function.

TABLE I
QUANTITATIVE RESULTS OF SENSING-AWARE Q FUNCTION.

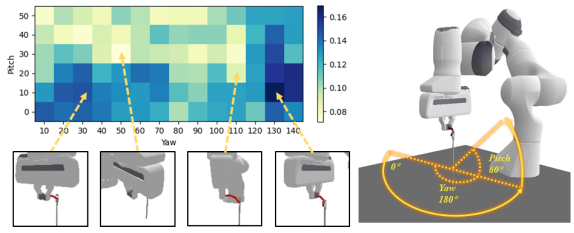
	Ours	*Ours(w/o pose-opt)*	*Fixed View*
Peg Insertion	**0.82**	0.64	0.71
Spatula Flipping	**0.65**	0.57	0.55
Needle Threading	**0.70**	0.46	0.66

2) Qualitative Visualization: We visualize the learned Q function (Fig. 6(a)) and the prediction error of the learned imitation learning policy (Fig. 6(b)) for the *Needle-Threading* task. We gather rendered images $\mathcal{I}^{sim}(\mathcal{P}^{c,i})$ from multiple view points and send them to the actor π^{sim} to get associated actions $\pi^{sim}(\mathcal{I}^{sim}(\mathcal{P}^{i}))$, and get the Q values $Q^{sim}(\mathcal{I}^{sim}(\mathcal{P}^{c,i}), \pi^{sim}(\mathcal{I}^{sim}(\mathcal{P}^{c,i})))$. It indicates that our Q function learns a reasonable policy to select the camera viewpoint.

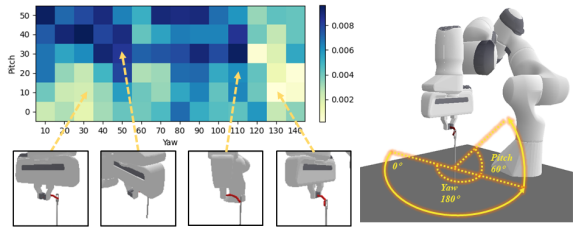
3) Salient Map of Actor Network: In order to have a better understanding about the actor network, we generate the salient map of the *Needle-Threading* task as shown in Fig. 7 . Given an input image denoted as $\mathcal{I}^{sim}$, we calculate the prediction error denoted as

$$\mathcal{L}_s = \|\hat{a}^{sim} - a^{sim}\|_2 \tag{26}$$

where $\hat{a}^{sim}$ and a^{sim} are the predicted action and ground-truth action, respectively. We visualize the gradient $\partial\mathcal{L}_s/\partial\mathcal{I}^{sim}$ as



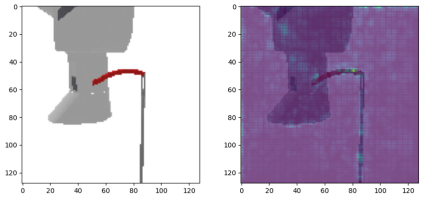
(a) Visualization of the learned Q function.



(b) Visualization of the the prediction error of the learned imitation learning policy.

Fig. 6. Qualitative visualization of sensing-aware Q function for the *Needle Threading*. Each pixel corresponds to a yaw and pitch value of the camera view.

the salient map. Fig. 7 indicates that the network pay more attentions to the thread's region, including the curved thread region and contact region between the thread and the needle.

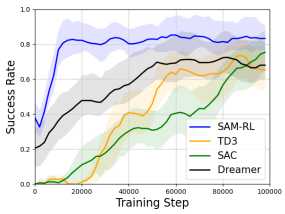


(a) *Input Image* (b) *Salient Map*

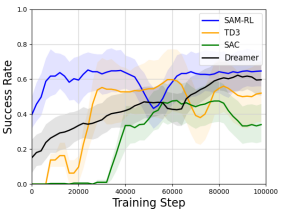
Fig. 7. The input image of the actor network and the corresponding salient map from the actor network

C. Comparing the SAM-RL with Model-free and Model-based Deep RL

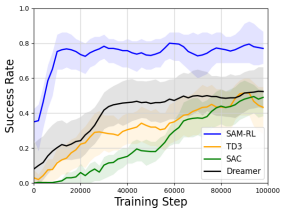
We compare our *SAM-RL* with common model-free deep RL algorithm *TD3* [56], *SAC* [58] and model-based deep RL algorithm *Dreamer* [6]. In this experiment, we adopt pybullet as the "real world". For *TD3*, *SAC*, and *Dreamer*, we directly train the robot in pybullet. The observation including an *RGB-XYZ* image and the position of the robot's end-effector. The action space is the translation of the robot's end-effector, which is 3 dimensions. The reward function of *Peg-Insertion* is the distance between the peg and the hole. The reward function of *Needle-Threading* is the distance between the thread and needle hole. The reward function of *Spatula-Flipping* is the position of the pancake among *z-axis* which is perpendicular to the ground. For *SAM-RL*, we develop the *model* via the differentiable physics-based simulation and rendering and use the *model* to complete tasks in pybullet ("real world"). Every time the environment is initialized or reset, the position and shape of the thread are randomly set within a certain range as described in Sec. IV-A. Fig. 8(a), Fig. 8(b), and Fig. 8(c) show the average success rate of the *SAM-RL*, *SAC*, *TD3*, and *Dreamer* during the training stage. Take *Needle-Threading* as an example, after training 100k steps, the average success rate of *SAC* models and *TD3* models are about 50%, while *Dreamer* is about 60%. However, our pipeline achieves a success rate of around 80% after 8k training steps. It indicates that our proposed *SAM-RL* is significantly sample-efficient compared to model-free deep reinforcement learning approaches and existing model-based method.



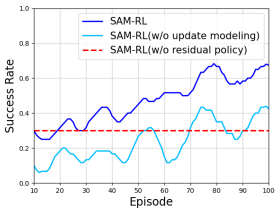
(a) Comparison with model-free and model-based RL in simulation on *Peg-Insertion*



(b) Comparison with model-free and model-based RL in simulation on *Spatula-Flipping*



(c) Comparison with model-free and model-based RL in simulation on *Needle-Threading*



(d) Ablation studies in real world

Fig. 8. Learning Curves of the *Peg-Insertion*, *Spatula-Flipping*, and *Needle Threading*. The *x-axis* shows the training steps/episodes and the *y-axis* indicates the success rate.

D. Ablation: Evaluating Components of SAM-RL

We apply *SAM-RL* in the real robot system and conduct two ablation study experiments.

- We remove the component of updating the *model* $\mathcal{M}$ by comparing $\mathcal{I}^{real}$ and $\mathcal{I}^{sim}$ and denote the experiment as *SAM-RL (w/o update modeling)*.
- We remove the residual policy used to address the sim2real gap and directly execute the predicted action a^{sim} in the real world denoted as *SAM-RL (w/o residual policy)*.

Note that *SAM-RL (w/o residual policy)* directly apply π^{sim} in real world, have no need to train, so it has a constant success rate. We post the result of *Needle Threading* in Fig. 8(d). It shows that these components play important roles in our pipeline, and removing these components results in significant performance decreases.

E. Evaluating the Model Updating via Differentiable Simulation and Rendering

1) Pose and Texture: With differentiable simulation and rendering, we can update the texture of the *model* $\mathcal{M}$. And also, we can update the pose of the *model* $\mathcal{M}$ during the manipulation. For *Peg-Insertion*, the *model*'s pose is the translation of the hole on the table, which is 2-dimension (x and y). We only need to update this at the beginning of each episode because the location of the hole is randomly initialized and fixed during manipulation. For *Spatula-Flipping*, The *model*'s pose is 5-dimension, include the translation of the pancake, which is 3-dimension (x, y, and z), and rotation of the pancake, which is 2-dimension (rotate among *x-axis* and *y-axis*) as the pancake is symmetry among the *z-axis*. For *Needle-Threading*, the thread is simulated using 10 links as described in Sec. IV-A. There are two revolute joints (See Fig. 5) between the links next to each other. So the *model*'s pose is the states of the joints, which is 20-dimension in total.

As shown in Fig. 9, we demonstrate the mean L_1 distance between the estimated and ground-truth position of the thread's links for *Needle-Threading*, and the L_1 distance between the estimated and ground-truth position of the pancake. To get the ground-truth position of the thread and pancake, we use the image rendered by Pybullet rendering, which is different from the differential rendering, as the ground-truth image. It indicate that our pipeline can reduce the distance between the estimated and the ground-truth state and texture by *model* updating via differentiable rendering. The first row shows the experiment where the thread (in the left most image) has a different position and different color from the ground truth image (as shown in the right most). The second row reflects the experiment where the thread has the same color but different position from the ground truth image. The last row shows the experiment where the pose of the pancake is different from the ground truth image.

Note that to better evaluate the performance of the *model* updating, we make the distance between the initialization and ground-truth larger. So it takes 200 iterations for *Needle-Threading* and 100 iterations for *Spatula-Flipping* to optimize. In practice, one action step can only cause a little change to the thread and pancake, so we only take 20 iterations for the thread updating and 10 iterations for the pancake updating at each action step.

2) Physical Attributes: Next we describe how to update objects' physical attributes (mass and inertial) in detail. We still use Pybullet to simulate the real world. We first use the spatula to push the pancake for T_1 time steps, then we stop moving the spatula and the pancake keeps sliding until time step T_2 (See Fig. 10). We use the robot to manipulate the spatula in torque control mode. So with the same robot action, the impulse provided by the robot is constant, and the pancake pose at time step T_2 is influenced by the mass value of the pancake. We update the object's physical attributes (mass and inertial) via the differentiable physics simulation. As shown in the learning curve, the relative error of the pancake's mass

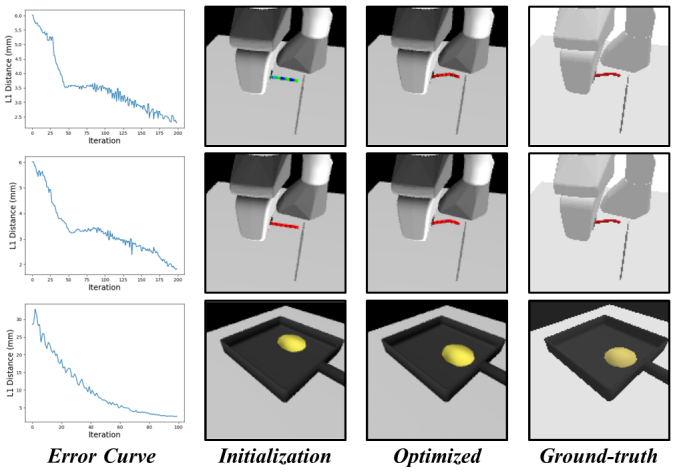


Fig. 9. The initialization and ground-truth image of the *model* updating for *Needle-Threading* and *Spatula-Flipping*. And the L_1 distance of the estimated and ground-truth state during *model* updating.

value can be reduced from 50% to around 20% after the optimization.

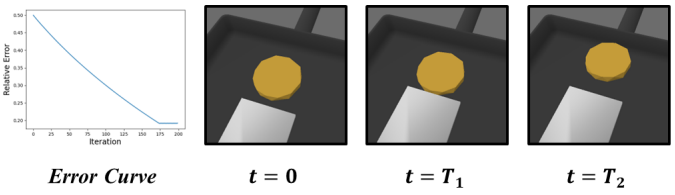


Fig. 10. At timestep $t = 0$, the robot start to push the pancake with the spatula until timestep $t = T_1$, then the robot stop moving and the pancake keep sliding until timestep $t = T_2$.

3) Real World: We also demonstrate the *model* updating via differentiable rendering in the real world. (See Fig. 11)

Fig. 11. The rightmost image is the real image, the target image for *model* update. The leftmost image and middle image show the initial state of the thread and the updated thread leveraging the differentiable rendering.

V. Conclusion

We propose a sensing-aware model-based reinforcement learning called *SAM-RL* leveraging the differentiable physics-based simulation and rendering. *SAM-RL* automatically updates the *model* by comparing the raw observations between simulation and the real world, and produces the policy efficiently. It also allows robots to select an informative viewpoint to better monitor the task process. We apply the system to

robotic assembly, tool manipulation, and deformable object manipulation tasks. Extensive experiments in the simulation and the real world demonstrate the effectiveness of our proposed learning framework.

VI. Acknowledgement

This work was in part supported by the National Key R&D Program of China (2021ZD0110700), Shanghai Municipal Science and Technology Major Project (2021SHZDZX0102), Shanghai Qi Zhi Institute, Shanghai Science and Technology Commission (21511101200), and a startup grant from National University of Singapore.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–503, 2016.

[3] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas *et al.*, "Solving rubik's cube with a robot hand," *arXiv preprint arXiv:1910.07113*, 2019.

[4] T. Wang, X. Bao, I. Clavera, J. Hoang, Y. Wen, E. Langlois, S. Zhang, G. Zhang, P. Abbeel, and J. Ba, "Benchmarking model-based reinforcement learning," *arXiv preprint arXiv:1907.02057*, 2019.

[5] M. Yang and O. Nachum, "Representation matters: offline pretraining for sequential decision making," in *International Conference on Machine Learning*. PMLR, 2021, pp. 11 784–11 794.

[6] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, "Dream to control: Learning behaviors by latent imagination," *arXiv preprint arXiv:1912.01603*, 2019.

[7] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, "Learning latent dynamics for planning from pixels," in *International conference on machine learning*. PMLR, 2019, pp. 2555–2565.

[8] D. Hafner, T. Lillicrap, M. Norouzi, and J. Ba, "Mastering atari with discrete world models," *arXiv preprint arXiv:2010.02193*, 2020.

[9] Y. Hu, L. Anderson, T.-M. Li, Q. Sun, N. Carr, J. Ragan-Kelley, and F. Durand, "Difftaichi: Differentiable programming for physical simulation," *ICLR*, 2020.

[10] K. Werling, D. Omens, J. Lee, I. Exarchos, and C. K. Liu, "Fast and feature-complete differentiable physics for articulated rigid bodies with contact," in *Proceedings of Robotics: Science and Systems (RSS)*, July 2021.

[11] T. A. Howell, S. Le Cleac'h, J. Z. Kolter, M. Schwager, and Z. Manchester, "Dojo: A Differentiable Simulator for Robotics," *Robotics: Science and Systems 2022*, Mar. 2022, under review. [Online]. Available: http://arxiv.org/abs/2203.00806

[12] T. Du, K. Wu, P. Ma, S. Wah, A. Spielberg, D. Rus, and W. Matusik, "Diffpd: Differentiable projective dynamics," *ACM Trans. Graph.*, vol. 41, no. 2, nov 2021. [Online]. Available: https://doi.org/10.1145/3490168

[13] Y. Li, T. Du, K. Wu, J. Xu, and W. Matusik, "Diffcloth: Differentiable cloth simulation with dry frictional contact," *ACM Trans. Graph.*, mar 2022, just Accepted. [Online]. Available: https://doi.org/10.1145/3527660

[14] Y. Qiao, J. Liang, V. Koltun, and M. Lin, "Differentiable simulation of soft multi-body systems," *Advances in Neural Information Processing Systems*, vol. 34, pp. 17 123–17 135, 2021.

[15] Y.-L. Qiao, J. Liang, V. Koltun, and M. C. Lin, "Scalable differentiable physics for learning and control," *arXiv preprint arXiv:2007.02168*, 2020.

[16] J. Liang, M. Lin, and V. Koltun, "Differentiable cloth simulation for inverse problems," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[17] S. Liu, T. Li, W. Chen, and H. Li, "Soft rasterizer: A differentiable renderer for image-based 3d reasoning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 7708–7717.

[18] T.-M. Li, M. Aittala, F. Durand, and J. Lehtinen, "Differentiable monte carlo ray tracing through edge sampling," *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, vol. 37, no. 6, pp. 222:1–222:11, 2018.

[19] C. Zhang, B. Miller, K. Yan, I. Gkioulekas, and S. Zhao, "Path-space differentiable rendering," *ACM Trans. Graph.*, vol. 39, no. 4, pp. 143:1–143:19, 2020.

[20] S. P. Bangaru, T.-M. Li, and F. Durand, "Unbiased warped-area sampling for differentiable rendering," *ACM Trans. Graph.*, vol. 39, no. 6, pp. 245:1–245:18, 2020.

[21] K. M. Jatavallabhula, M. Macklin, F. Golemo, V. Voleti, L. Petrini, M. Weiss, B. Considine, J. Parent-Levesque, K. Xie, K. Erleben, L. Paull, F. Shkurti, D. Nowrouzezahrai, and S. Fidler, "gradsim: Differentiable simulation for system identification and visuomotor control," *International Conference on Learning Representations (ICLR)*, 2021. [Online]. Available: https://openreview.net/forum?id=c_E8kFWfhp0

[22] P. Ma, T. Du, J. B. Tenenbaum, W. Matusik, and C. Gan, "Risp: Rendering-invariant state predictor with differentiable simulation and rendering for cross-domain parameter estimation," in *International Conference on Learning Representations*, 2021.

[23] J. Lv, Q. Yu, L. Shao, W. Liu, W. Xu, and C. Lu, "Sagci-system: Towards sample-efficient, generalizable, compositional, and incremental robot learning," in *2022 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2022.

[24] R. ROS, "urdf," http://wiki.ros.org/urdf, 2020.

[25] F. R. Inc., "Flexiv robot arm," 2019. [Online]. Available: https://www.flexiv.com/en/application

[26] F.-M. Luo, T. Xu, H. Lai, X.-H. Chen, W. Zhang, and Y. Yu, "A survey on model-based reinforcement learning," *arXiv preprint arXiv:2206.09328*, 2022.

[27] C. Collander, W. J. Beksi, and M. Huber, "Learning the next best view for 3d point clouds via topological features," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 12 207–12 213.

[28] M. Krainin, B. Curless, and D. Fox, "Autonomous generation of complete 3d object models using next best view manipulation planning," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 5031–5037.

[29] D. Peralta, J. Casimiro, A. M. Nilles, J. A. Aguilar, R. Atienza, and R. Cajote, "Next-best view policy for 3d reconstruction," in *European Conference on Computer Vision*. Springer, 2020, pp. 558–573.

[30] S. Kriegel, C. Rink, T. Bodenmüller, and M. Suppa, "Efficient next-best-scan planning for autonomous 3d surface reconstruction of unknown objects," *Journal of Real-Time Image Processing*, vol. 10, no. 4, pp. 611–631, 2015.

[31] Z. Wu, S. Song, A. Khosla, X. Tang, and J. Xiao, "3d shapenets for 2.5 d object recognition and next-best-view prediction," *arXiv preprint arXiv:1406.5670*, vol. 2, no. 4, 2014.

[32] Y. Han, I. H. Zhan, W. Zhao, and Y.-J. Liu, "A double branch

next-best-view network and novel robot system for active object reconstruction," in *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 7306–7312.

[33] T. Foissotte, O. Stasse, A. Escande, P.-B. Wieber, and A. Kheddar, "A two-steps next-best-view algorithm for autonomous 3d object modeling by a humanoid robot," in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 1159–1164.

[34] B. Browatzki, V. Tikhanoff, G. Metta, H. H. Bülthoff, and C. Wallraven, "Active in-hand object recognition on a humanoid robot," *IEEE Transactions on Robotics*, vol. 30, no. 5, pp. 1260–1269, 2014.

[35] S. Kriegel, T. Bodenmüller, M. Suppa, and G. Hirzinger, "A surface-based next-best-view approach for automated 3d model completion of unknown objects," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 4869–4874.

[36] H. Surmann, A. Nüchter, and J. Hertzberg, "An autonomous mobile robot with a 3d laser range finder for 3d exploration and digitalization of indoor environments," *Robotics and Autonomous Systems*, vol. 45, no. 3, pp. 181–198, 2003.

[37] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart, "Receding horizon "next-best-view" planner for 3d exploration," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 1462–1468.

[38] H. Kato, D. Beker, M. Morariu, T. Ando, T. Matsuoka, W. Kehl, and A. Gaidon, "Differentiable rendering: A survey," *arXiv preprint arXiv:2006.12057*, 2020.

[39] S. Zhao, W. Jakob, and T.-M. Li, "Physics-based differentiable rendering: from theory to implementation," in *ACM siggraph 2020 courses*, 2020, pp. 1–30.

[40] P. Sundaresan, R. Antonova, and J. Bohg, "Diffcloud: Real-to-sim from point clouds with differentiable simulation and rendering of deformable objects," *arXiv preprint arXiv:2204.03139*, 2022.

[41] J. Luo, E. Solowjow, C. Wen, J. A. Ojea, A. M. Agogino, A. Tamar, and P. Abbeel, "Reinforcement learning on variable impedance controller for high-precision robotic assembly," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 3080–3087.

[42] M. A. Lee, Y. Zhu, K. Srinivasan, P. Shah, S. Savarese, L. Fei-Fei, A. Garg, and J. Bohg, "Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8943–8950.

[43] L. Shao, T. Migimatsu, and J. Bohg, "Learning to scaffold the development of robotic manipulation skills," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 5671–5677.

[44] J. Xu, Z. Hou, Z. Liu, and H. Qiao, "Compare contact model-based control and contact model-free learning: A survey of robotic peg-in-hole assembly strategies," *arXiv preprint arXiv:1904.05240*, 2019.

[45] Y. Chebotar, O. Kroemer, and J. Peters, "Learning robot tactile sensing for object manipulation," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 3368–3375.

[46] T. Tsuji, J. Ohkuma, and S. Sakaino, "Dynamic object manipulation considering contact condition of robot with tool," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 3, pp. 1972–1980, 2016.

[47] J. Silvério, G. Clivaz, and S. Calinon, "A laser-based dual-arm system for precise control of collaborative robots," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 9183–9189.

[48] S. Huang, Y. Yamakawa, T. Senoo, and M. Ishikawa, "Robotic needle threading manipulation based on high-speed motion strategy using high-speed visual feedback," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 4041–4046.

[49] H. Kim, Y. Ohmura, and Y. Kuniyoshi, "Gaze-based dual resolution deep imitation learning for high-precision dexterous robot manipulation," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1630–1637, 2021.

[50] A. Dai, M. Nießner, M. Zollöfer, S. Izadi, and C. Theobalt, "Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface re-integration," *ACM Transactions on Graphics 2017 (TOG)*, 2017.

[51] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon, "Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera," in *UIST '11 Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, October 2011, pp. 559–568.

[52] G. Thomas, M. Chien, A. Tamar, J. A. Ojea, and P. Abbeel, "Learning robotic assembly from cad," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 3524–3531.

[53] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2980–2988.

[54] H. Fan, H. Su, and L. J. Guibas, "A point set generation network for 3d object reconstruction from a single image," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 605–613.

[55] T. Silver, K. Allen, J. Tenenbaum, and L. Kaelbling, "Residual policy learning," *arXiv preprint arXiv:1812.06298*, 2018.

[56] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*. PMLR, 2018, pp. 1587–1596.

[57] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2021.

[58] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.