

Supplementary Materials

Anonymous Author(s)

Affiliation

Address

email

1 More Discussions on Real2Sim

2 1.1 Why not use video tracking?

3 One may be curious about why not use video tracking to extract a tie’s motion in our work. We test
4 co-tracker [1] and DINO-Tracker [2] on our tie-knotting demonstration videos. Please move to the
5 folder ”./video tracking results” to see example outputs. We can find that even the state-of-the-art
6 video tracking model cannot track accurate tying motion.

7 1.2 Experiment on out-of-distribution issue of keypoints detection

8 To illustrate the necessity of iteratively updating the detection model, we train a neural network with
9 the same structure on a randomly sampled tie’s shape based on the initial shape. We trained our
10 iterative keypoint detection model on 14 different shapes separately. For each shape, we randomly
11 generate 500 similar shapes for training. For the random sampled method, we randomly generate
12 7000 shapes from the initial tie’s shape to train a single keypoint detection model. We compare
13 the result of **Ours** with random sample **RS** result in Fig. 1. Compared to the human annotation
14 result, **RS**’s predictions have larger errors than **Ours**. **RS** method encounters an out-of-distribution
15 problem. The test shape of the tie cannot be easily sampled, so **RS** cannot generalize to this test
16 case.

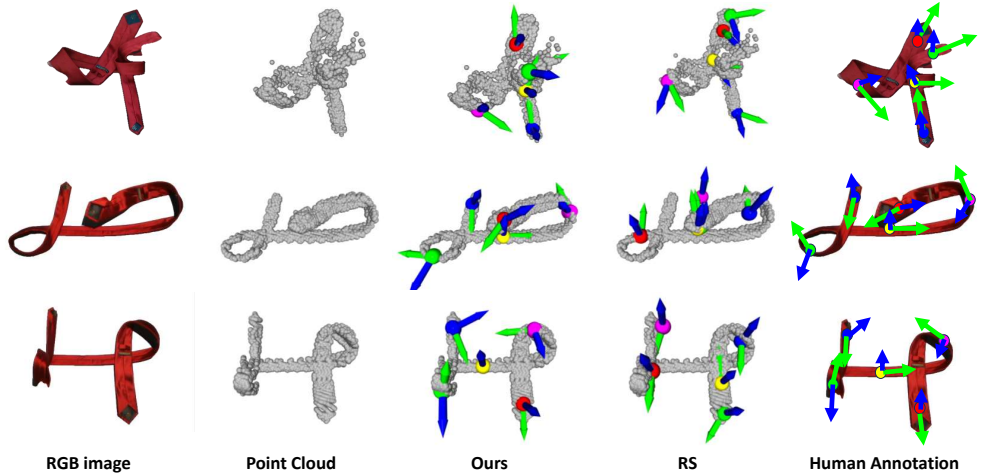


Figure 1: Prediction results of the oriented keypoints on real image and point cloud.

17 We also test the iterative global keypoint detection and **RS** method in simulation with ground truth
18 annotation. The quantitative results are listed in Tab. 1.

Table 1: Quantitative results of iterative keypoint detection on simulation data

	Position Error(m)	Z-axis Error(°)	X-axis Error(°)
<i>Ours</i>	0.028	10.68	14.41
<i>RS</i>	0.183	49.00	68.76

1.3 Implementation details of keypoints detection

1.3.1 keypoint positions prediction

1) data generation We first load the mesh model into *DiffClothAI* [3], choose keypoint as control vertices, and apply random perturbations to these vertices to generate different shapes of the tie. Then, we load generated mesh models into *PyBullet* [4] to render point clouds. We use the same camera intrinsics as demonstrated video and similar camera pose to render point clouds. We generate 500 point clouds as training data.

To annotate training data, we first compute the geodesic distance of all points in the point cloud to keypoints. Then, we convert the geodesic distance to probability using equation 1. d is the geodesic distance, σ is a hyperparameter, p is the probability to evaluate how likely the point is to be a keypoint. In our experiments, we use $\sigma = 0.15$. Thus, if a point is close to one of five keypoints, the probability corresponding to that keypoint will be high.

$$p = \exp^{-\frac{d^2}{2\sigma^2}} \quad (1)$$

2) training details Different from the original pointnet++ semantic segmentation model [5], we change the last layer to sigmoid. The training parameters are listed in Tab. 2.

parameter name	parameter value
loss function	L2(for keypoint positions and offsets prediction) L1(for normal and middle line direction prediction)
data augmentation	gaussian noise, random scale, random rotation
training epochs	80
batch size	24
learning rate	1e-4
optimizer	Adam
scheduler	cosine annealing with 10 epochs warm-up

Table 2: Hyperparameters for training global keypoint prediction

3) inference details Our model takes a point cloud as input and outputs a probability matrix $P \in (0, 1)^{N \times 5}$, N is the number of points in the point cloud. Each entry $P_{i,j}$ represents the probability of point i to be keypoint j . To decode the predicted keypoints positions, we first select points with the top 5% probability as inlier for each column of P . Then, we assign other points' probability to zero and normalize the probability for each column of P . Now we get the normalized probability distribution of each keypoint, denoted as \hat{P} . Finally, we compute the average positions of all points weighted by normalized probability, $x_k = \sum_{i=1}^N \hat{P}_{i,k} \cdot x_i$. This is the final prediction for keypoint positions.

1.3.2 normal(z axis) prediction

1) data generation We generate data the same way as keypoint positions. For annotation, we first compute the normal direction of each face of the mesh. Then, we assign these values to points in the point cloud according to the nearest faces.

2) training details We also remove the log_softmax layer in pointnet++ [5]. We use L1 distance as the loss function. The other training parameters are the same as keypoint position prediction.

47 **3) inference details** With predicted keypoints positions and predicted normal directions of all
48 points, we compute the normal of each keypoint as the average of neighboring points normal direc-
49 tions.

50 **1.3.3 middle line(x axis) prediction**

51 **1) data generation** Same as normal prediction, just change the annotation from normal direction
52 to middle line direction.

53 **2) training details** It’s the same as normal prediction.

54 **3) inference details** It’s the same as normal prediction.

55 **1.4 Results of local feature matching and keypoints detection on real data**

56 We present some examples of local feature matching and keypoints detection on two tie-knotting
57 tasks and a towel-folding task, shown in Fig 2.

58 **1.5 Ablation study of HFM on real data**

59 We demonstrate the effectiveness of hierarchical matching in cloth state estimation in Fig. 3. In the
60 first test case, we aim to illustrate the importance of global keypoint detection. Therefore, we chose
61 two images that show differences in positions and orientations. In the third column, **Ours** method
62 successfully flips the tie and moves forward a little, as shown in the images. **Our w/o KP** and **Ours**
63 **w/o LF** cannot move forward as expected. Because, in this case, local feature matching cannot find
64 correspondences in the tie’s left part.

65 In the second test case, we aim to illustrate the importance of local feature matching. We chose two
66 images that contain an operation of lifting a side of a ring in the air. This action requires detailed
67 information to achieve accurate estimation. The last column shows the result of **Ours w/o FM**.
68 Our framework cannot accurately estimate the shape only with global keypoint positions and local
69 frames. This global information can only provide general structure guidance instead of detailed
70 shape information.

71 **2 More discussions on Learn@sim**

72 **2.1 How to control tie in *DiffClothAI***

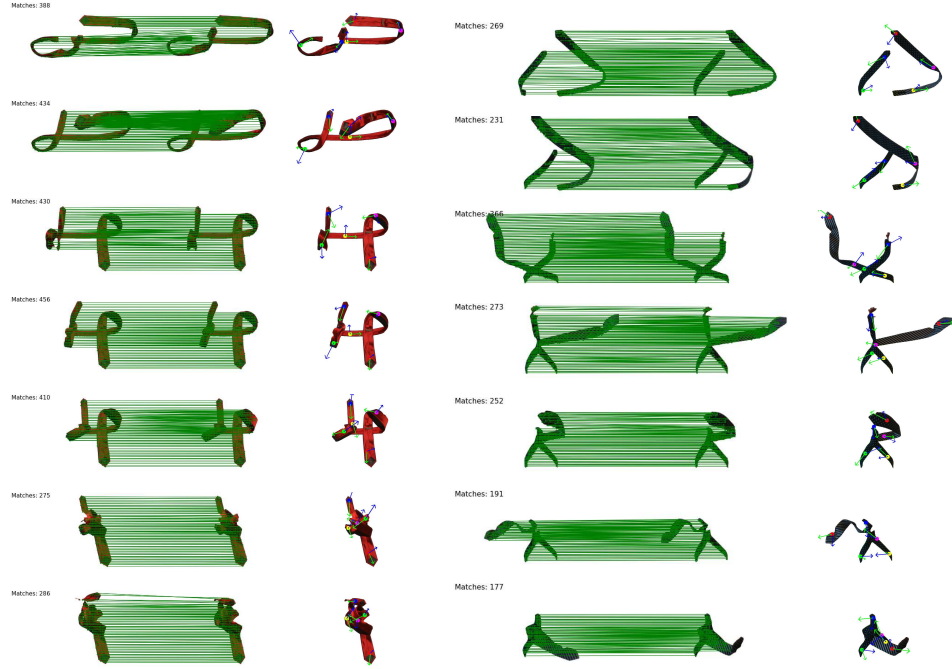
73 Modeling grasping in *DiffClothAI* [3] is not simply selecting one vertex on the mesh as the control
74 vertex. Because controlling one vertex is not enough to simulate rotation in *DiffClothAI*, knotting
75 a tie requires some rotation actions. Therefore, we select one central vertex and its surrounding
76 vertices as control vertices, shown in Fig.4. By controlling a small region instead of a single vertex,
77 we can simulate rotation actions in *DiffClothAI*.

78 **2.2 Implementation details of teach-student training paradigm**

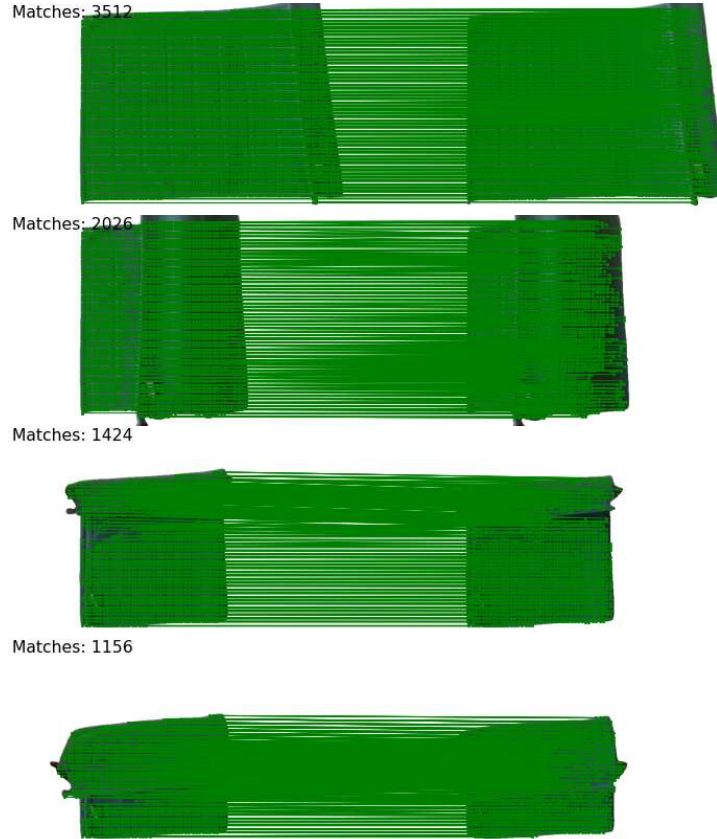
79 **2.2.1 Teacher policy**

80 We model the grasping point selection as MDP and use model-free RL to learn the proper grasping
81 point. To simplify the problem, we sample 40 vertices on the middle line of the mesh model as our
82 candidates. vertices directly connected to these candidates in left, right, up, and down are defined as
83 their neighbors.

84 In practice, we evenly sample 40 vertices on the middle line of the mesh model as our candidates.
85 The state s is a 40×6 matrix. The action a is a 820×1 one-hot vector, which contains grasping
86 one vertex(40) and two vertices($40 \times 39/2 = 780$).

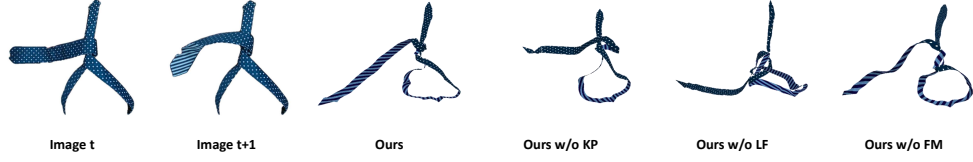


(a) Local feature matching and keypoints (b) Local feature matching and keypoints detection re-detection results on real-world tie-knotting sults on another real-world tie-knotting demonstration demonstration

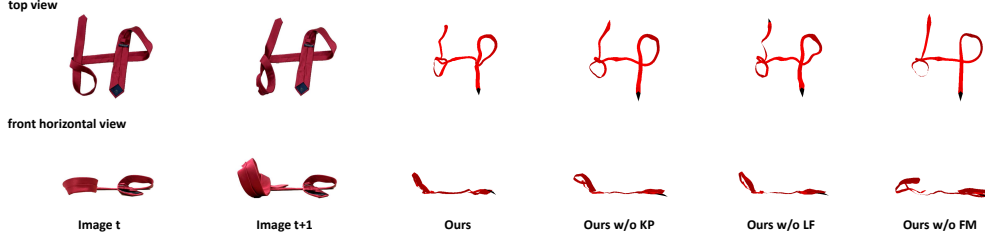


(c) Local feature matching results on another real-world towel-folding demonstration

Figure 2: We test local feature matching and keypoints detection on real-world demonstrations. It shows that our method works for most tie shapes.



(a) Ablation study mainly on keypoint prediction.



(b) Ablation study mainly on local feature matching.

Figure 3: Ablation study on hierarchical feature matching for state estimation.

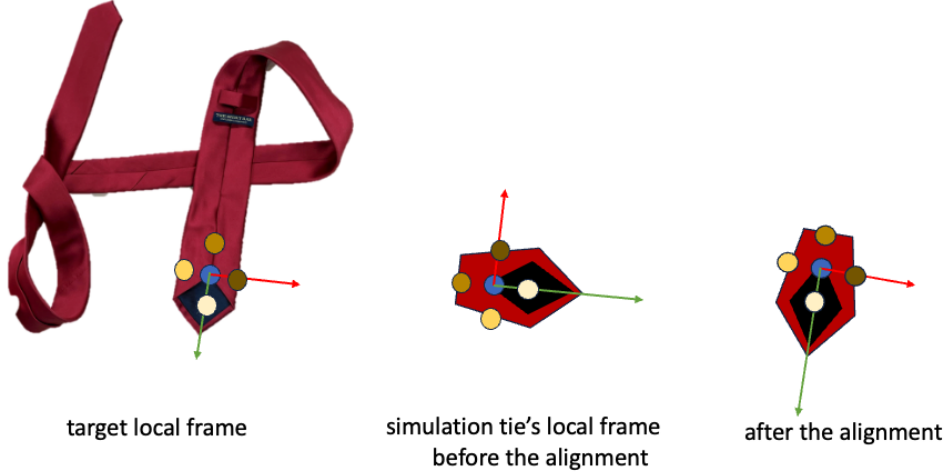


Figure 4: Illustration of control vertices in *DiffClothAI*.

87 To learn to select grasping points, we use PPO implemented in [stable-baseline3](#) [6]. The hyperpa-
 88 rameters are shown in Tab 3 for all trajectories.

89 2.2.2 Student policy

90 Our student policy learns to predict grasping point positions from the point cloud. The training
 91 details are the same as training keypoints prediction, only changing the number of keypoints from
 92 5 to 2. We follow the same training hyperparameters as grasping point prediction for placing point
 93 position detection, only changing the pointnet++ semantic segmentation model to the classification
 94 model.

95 2.3 More results on ATM baseline

96 We first illustrate example outputs of ATM baseline on two different ties in Fig. 5. We can see that
 97 without explicit mesh modeling, ATM will quickly deviate from correct trajectories.

parameter name	parameter value
learning rate	0.0003
batch size	64
γ	0.99
gae_lambda	0.95
clip range	0.2
C_1	5
fitting threshold	{0.9, 1.5, 2.0, 3.0, 3.0, 1.9} (listed in subgoals order)
C_2	30
C_3	30

Table 3: Hyperparameters for learning grasping points settings

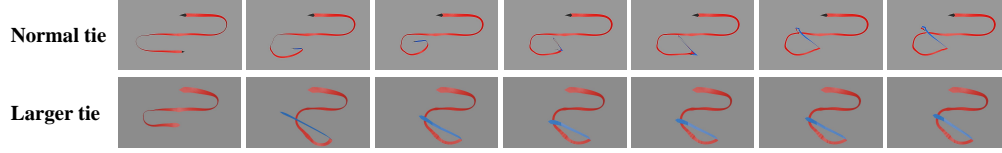


Figure 5: Illustration of ATM rollouts in simulation.

98 To further examine whether the long-horizon property or points trajectories representations lead to
 99 the failure of ATM on tie-knotting tasks, we further conduct experiments on some shorter tasks to
 100 see if ATM can work. Specifically, we divide the whole tie-knotting task into 6 subtasks, training
 101 and testing ATM on each subtask separately. The results are shown in Fig. 6. We can see that ATM's
 102 results look better for the first 3 subtasks. But ATM still cannot complete the last 3 subtasks, which
 103 involve complex topology and subtle dynamics. This experiment demonstrates that using points
 104 trajectory representation cannot handle such complex tasks even with a shorter horizon.

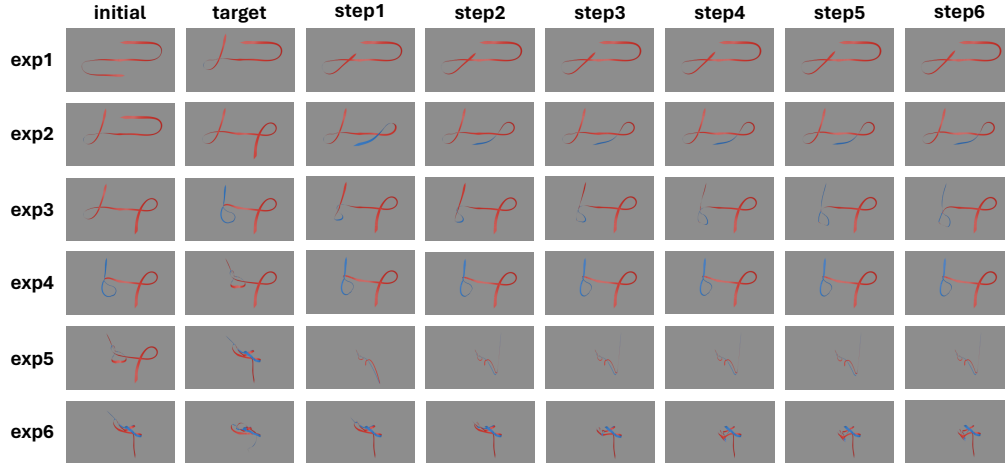


Figure 6: Illustration of ATM results on 6 subtasks.

105 3 Real world experiments

106 3.1 Experiment setup

107 We set up the real-world experiment with a dual-arm robot as shown in Fig. 7. The MOVO robot [7]
 108 has two 7 DoF arms and a Kinect RGB-D camera overhead. We perform position controls and use
 109 RangedIK [8] for solving inverse kinematics. The success state is defined in Fig. 8.

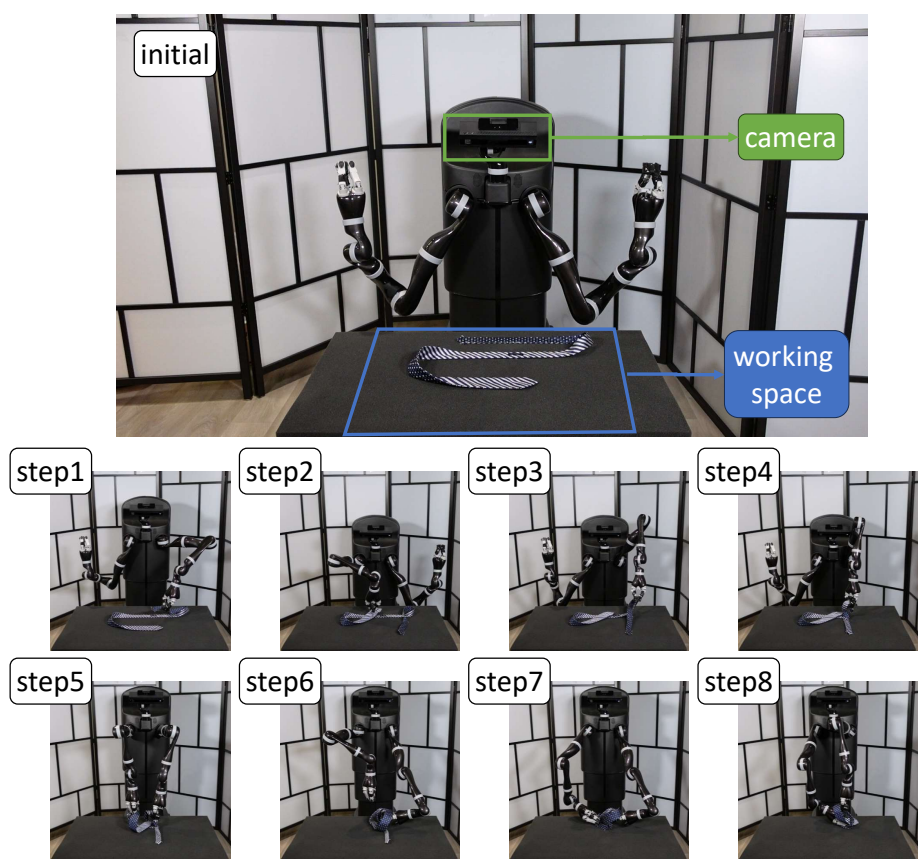


Figure 7: Illustration of real-world experiment settings.

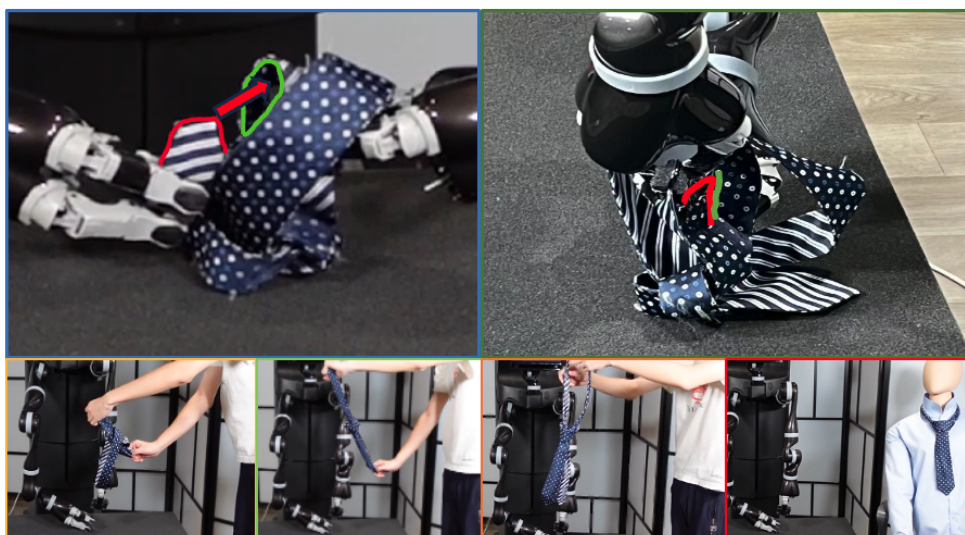


Figure 8: Illustration of the success state of knotting a tie.

110 3.2 Failure cases and analysis

111 Two major failure cases in real-world experiments are shown in Fig. 9. One is the robot fails to
112 rotate the whole ring structure of the tie, another is the robot fails to insert the little end of the tie
113 into the whole shown in Fig. 8.

114 The first case is caused by the subtle dynamics of the tie. To rotate the ring structure, the tie should
115 be a bit harder so that the ring structure will not crumple during the rotation process, while it should
116 not be so hard so that the rotation action won't interfere with other parts of the tie. This places a
117 high demand on both the tie and the robot. It's hard to solve from the algorithm side.

118 The second case is caused by partial observation of this task. We use one camera on the top of the
119 robot for perception. It cannot perceive the little end of the tie in this case. Thus, the robot has to act
120 blindly, lowering the success rate.

121 For more fail cases, please move to the folder "/fail cases".

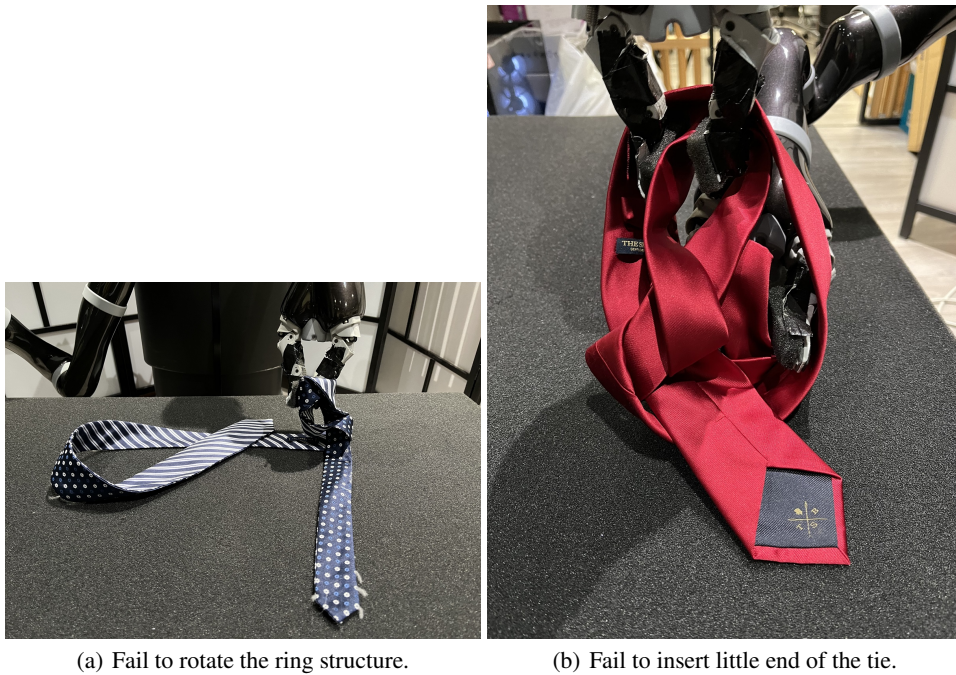


Figure 9: Illustration of two major failure cases

References

- [1] N. Karaev, I. Rocco, B. Graham, N. Neverova, A. Vedaldi, and C. Rupprecht. CoTracker: It is better to track together. 2023.
- [2] N. Tumanyan, A. Singer, S. Bagon, and T. Dekel. Dino-tracker: Taming dino for self-supervised point tracking in a single video, March 2024.
- [3] X. Yu, S. Zhao, S. Luo, G. Yang, and L. Shao. Diffclothai: Differentiable cloth simulation with intersection-free frictional contact and differentiable two-way coupling with articulated rigid bodies. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023.
- [4] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. <http://pybullet.org>, 2016–2021.
- [5] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 2017.
- [6] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22 (268):1–8, 2021. URL <http://jmlr.org/papers/v22/20-1364.html>.
- [7] Kinova. Kinova-movo. URL https://docs.kinovarobotics.com/kinova-movo/Concepts/c_movo_hardware_overview.html.
- [8] Y. Wang, P. Praveena, D. Rakita, and M. Gleicher. Rangedik: An optimization-based robot motion generation method for ranged-goal tasks. *arXiv preprint arXiv:2302.13935*, 2023.