2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Goal-Conditioned Action Space Reduction for Deformable Object Manipulation

Shengyin Wang[1] Rafael Papallas[1] Matteo Leonetti[2] Mehmet Dogar[1]

Abstract— **Planning for deformable object manipulation has been a challenge for a long time in robotics due to its high computational cost. In this work, we propose to reduce this cost by reducing the number of pick points on a deformable object in the action space. We do this by identifying a small number of key particles that are sufficient as pick points to reach a given goal state. We find these key particles through a geometric model simplification process, which finds the minimal geometric model that still enables a good approximation of the original model at the goal state. We present an implementation of this general approach for 1-D linear deformable objects (e.g., ropes) that uses a piece-wise line fitted model, and for 2-D flat deformable objects (e.g., cloth) that uses a mesh simplified model. We conducted simulation experiments on ropes and cloths, which demonstrate the effectiveness of the proposed method. Finally, the planned paths are executed in a real-world setting for two cloth folding tasks.**

I. INTRODUCTION

Consider a scenario where a robot is required to manipulate a piece of cloth into a specific goal configuration, like the object folded diagonally in Fig. 1. A general way to achieve this goal is building a model of the object in simulation, and optimizing a trajectory within a planning framework. Deformable objects are usually modeled as mass-spring systems, with each mass point (which we refer to as a *particle* in this work) also representing a possible pick point for the robot. High-fidelity models of deformable objects often contain thousands of such particles, making planning extremely computationally expensive.

In this work, we propose to reduce this computational cost by reducing the number of pick points on the object. We do this by identifying a small number of *key particles* that are sufficient as pick points to reach the goal state. For example, for the object in Fig. 1, we identify four key particles as sufficient. We find these key particles through a *geometric model simplification* process, which finds the minimal geometric model that still enables a good approximation of the original model at the goal state. For example, in Fig. 1, the simplified mesh model shown in the third frame provides a good approximation of the original geometric model at the goal state, shown in the second frame. Planning using the resulting small set of key particles in the action space is then significantly faster, as opposed to planning using the original model with several thousands of particles.

Deformable object manipulation has received significant interest from robotic researchers in recent years because of the ubiquitous existence of deformable objects in our daily life and their extensive applications in both domestic [1], [2] and industrial scenarios [3]. While impressive progress has been achieved for rigid object manipulation, the development of deformable object manipulation methods lags behind particularly due to two main challenges: high-dimensional state representation and complex dynamics. These challenges make both perception and planning complex problems, especially for traditional search or optimization based methods [4]–[6]. Detailed surveys of deformable modeling, planning, control, and learning can be found in Zhu et al. [7], Yin et al. [8], Bhagat et al. [9], and Arriola-Rios et al. [10].

The idea of identifying important points/features on deformable objects has been studied before. One way of approaching this problem is to determine particular features for a specific task. For example, Sun et al. [11] detect and eliminate wrinkles, where the task is to flatten a deformable object. Other approaches use manually input key points [12], or contours [13]. Recently, learning-based methods for deformable object manipulation have been developed [14]–[21]. Some learning approaches aim to simplify deformable object representations and identify key features on them. For example, Yan et al. [15] propose to extract a compact representation of the deformable object directly from raw sensor inputs for dynamics learning and achieve faster planning. Other works are dedicated to learning deformable dynamics using graph neural networks [18] based on extracted key points or the underlying mesh representations [20]. Power et al. [22] use simple models as a cheaper data collection way to improve learning efficiency.

In this work, we also exploit the idea of simplifying deformable object models and identifying important features on them to reduce the action space. However, our approach differs from the above in that our method (i) can adapt to different tasks (as opposed to identifying features for one specific task); (ii) identifies the key particles autonomously; and (iii) takes a model-based (as opposed to learning-based) *and* goal-conditioned approach. These enable our method to accept arbitrary goal configurations of the object and to quickly perform simplification of the object model and reduction of the action space conditioned on this goal, without requiring goal-specific training.

In particular, we contribute:

- A general action space reduction scheme integrated

This research has received funding from the UK Engineering and Physical Sciences Research Council under the grant EP/V052659/1.

 Code & data available at https://github.com/shengyin-git/ac_softagent.

[1]S. Wang, R. Papallas and M. Dogar are with the School of Computing, University of Leeds, UK. {scswan, r.papallas, m.r.dogar}@leeds.ac.uk

[2]M. Leonetti is with the Dept. of Informatics, King's College London, UK. matteo.leonetti@kcl.ac.uk

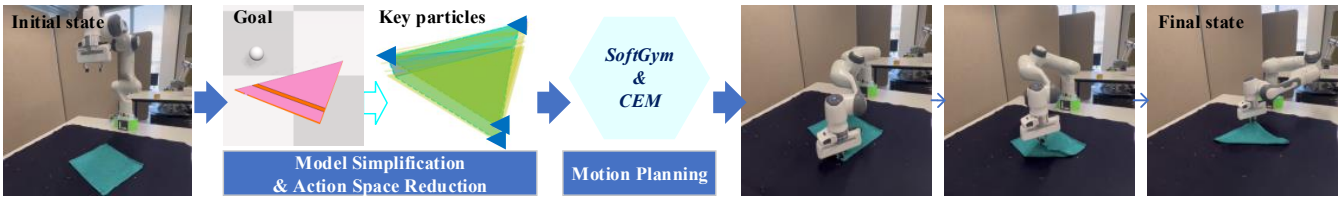

Fig. 1. Overview of the proposed approach. Given the task of folding a piece of cloth diagonally, our approach simplifies the geometric model at the goal configuration to extract the key particles as denoted by blue triangles, which represent the graspable points in the reduced action space. After that, Cross Entropy Method (CEM) is implemented to search a folding plan in the SoftGym simulator. A Franka Panda robot executes the plan in the real world and achieves a goal state.

into a manipulation planning pipeline based on goal-conditioned geometric model simplification for deformable objects (Sec. III-A);

- An implementation of this general approach for 1-D linear deformable objects (e.g., ropes) that uses a piece-wise line fitted model (Sec. III-B), and for 2-D flat deformable objects (e.g., cloth) that uses a mesh simplified model (Sec. III-C);
- An extension of this approach to multi-step planning with intermediate goal configurations, where geometric model simplification and action space reduction are performed on successive intermediate goals (Sec. III-D).

We perform a set of simulated experiments with rope and cloth objects, where we investigate if a planner using the reduced action space with autonomously extracted key particles can reach the goal state more efficiently and more successfully than a planner using the original action space (Sec. IV-C). We also present example demonstrations of key particle based manipulation plans on a real robot system (Sec. IV-D).

II. PROBLEM FORMULATION

We consider the planning problem of manipulating a deformable object into a given goal configuration. While manipulation happens in 3-D space, we focus on objects that can be represented by 1-D lines (like ropes) or 2-D surfaces (like cloths). For example, straightening a crumpled rope (Fig. 7-(a)) or folding a piece of cloth diagonally into half (Fig. 1). In this section, we define the state space, action space, and objective function of the manipulation problem.

For both 1-D linear and 2-D flat deformable objects modeled by mass-spring systems, the geometric model can be represented as a connected undirected graph $M = (V, E)$, where $V = \{1, 2, ..., N\}$ denotes a set of particles indexed from 1 to $N = |V|$, and $E \subseteq \{\langle i, j\rangle \,|\, i, j \in V \; and \; i \neq j\}$ denotes the edges. During manipulation, the configuration varies while the geometric model remains unchanged.

To describe the configuration of the deformable model at each time step t during manipulation, we use the positions of all the particles as the current state of the object, which is denoted by $\xi_t := \{p_i \,|\, \forall i \in V\}$ where $p_i = [x_i, y_i, z_i]$ represents the position of the i^{th} particle.

As for action space A, we assume the robot is equipped with one picker, which can pick any given particle in the object, that is, $\{\forall i \in V\}$, and move it by a certain distance along a direction in 3D space. We also add a $None$ action, corresponding to not holding any picking point. The action at time t can be represented as:

$$a_t = \begin{cases} \langle i, \delta x_t, \delta y_t, \delta z_t \rangle \\ None \end{cases} \tag{1}$$

Furthermore, we constrain the movement of the picker at each step to avoid moving the deformable object drastically: $\|\delta x_t\| \leq \Delta x$, $\|\delta y_t\| \leq \Delta y$, $\|\delta z_t\| \leq \Delta z$; Δx, Δy and Δz are motion limits along each axis in Cartesian space.

For the dynamics model, we use the one provided by SoftGym [23], which is built on top of PyFlex [24]. Within the simulator, ropes and cloths are modeled with a sequence of particles or a grid of particles respectively. Neighboring particles are connected by stretching constrained springs while one-step-away particles are constrained by bending springs. Meanwhile, self-collision of all particles is also considered, details of which can be found in [24].

We formulate the manipulation problem as a finite-horizon planning problem, whose solution is a sequence of T actions, $\tau = \langle a_1, a_2, ..., a_T \rangle$, that minimizes the distance between the final configuration of the particles and their goal configuration ξ_G:

$$\begin{aligned} \min_{\tau} \quad & \|\xi_G - \xi_T(\tau)\| \\ s.t. \quad & C(a_t) < 0, \; \forall t \in \{1, ..., T\}, \end{aligned} \tag{2}$$

where $C(a_t)$ is used to eliminate actions that violate model and environmental constraints, such as stretching and bending limits, or working space restrictions at step t, which are defined in the SoftGym simulator.

In this planning framework, the size of the action space is $|A| = (N \times \mathbb{R}^3 + 1)$. The number of graspable particles affects linearly the size of the action space, which has, in turn, an exponential effect on the search space through the branching factor. Therefore, the number of particles considered for planning has a significant impact on planning efficiency. In the next section we present a methodology to minimize it.

III. METHODOLOGY

We propose to reduce the action space by simplifying the original geometric model M_O to $M_S = (V_S, E_S)$, with a smaller number of particles $N_S = |V_S|$. The particles are computed from the goal configuration as described in this section. The method we use to simplify the geometric model differs for objects that can be approximated with 1-D models and objects that can be approximated with 2-D models. In Sec. III-A, we summarize the overall approach, in Sec. III-B we focus on 1-D models and in section Sec. III-C on 2-D models. Lastly, we describe how our planner can be applied

to tasks for which the goal configuration is not enough to obtain all the necessary key particles for planning to it from the initial configuration. Such cases can be tackled with multi-step planning to intermediate goal configurations, as described in Sec. III-D.

A. Action Space Reduction

The overall algorithm for action space reduction is illustrated in Alg. 1. The inputs of the method are the original geometric model M_O and goal configuration of the original model ξ_G^O, while the outputs are the simplified geometric model M_S and the extracted key particle set $\hat{V}_O \subseteq V_O$ in the action space, which is related yet different from the set of particles V_S of the simplified model.

The algorithm starts by reducing the original geometric model to the simplest form, in which the number of target simplification elements is 2 (line 1). Given the original geometric model, the goal configuration of the model, and the target number of the model elements, the simplification function is invoked to reduce the original geometric model M_O to M_S, with corresponding simplified goal configuration ξ_G^S (line 3).

Then, the distance between the goal configurations of the original geometric model and the simplified model is calculated as the *error*, indicating how good a fit the simplified geometric model is to the original model, at the goal (line 4). If the error is below a threshold (a parameter set by the user), the simplification process is terminated (line 6); if not, the simplification process repeats with a more complex simplified geometric model (line 5). Finally, we extract $\hat{V}_O$, which includes the key particles in the original geometric model that correspond to the particles of the simplified geometric model V_S (line 7). $\hat{V}_O$ is the output and is used to construct the reduced action space for the original object.

Algorithm 1: Action Space Reduction Process

Input: M_O, ξ_G
Output: M_S, $\hat{V}_O$
1 $N_S = 2$;
2 **do**
3 $\quad M_S, \xi_G^S \leftarrow$ `Simplify`(M_O, ξ_G^O, N_S);
4 $\quad error \leftarrow$ `Error`(ξ_G^S, ξ_G^O);
5 $\quad N_S \leftarrow N_S + 1$;
6 **while** *error* > *threshold*;
7 $\hat{V}_O \leftarrow$ `Extract`(M_S, ξ_G^S, M_O, ξ_G^O)

B. Piece-wise Line Fitting for 1-D Linear Models

For 1-D linear models, we adopt piece-wise line fitting method to implement the `Simplify` function (Alg. 1, line 3). Given the number of key particles N_S of the target simplification, a Quadratic Programming (QP) problem is defined and solved to find the optimal position of the N_S particles. The cost function of the QP problem to be minimized is the "distance" between the fitted piece-wise lines and the original shape of the object. To find such a distance value, we evenly sample the same number of points N_E on the two models. For example, in Fig. 2, $N_E = N_O$ points, represented by yellow dots and grey dots, are sampled respectively. We find the mean distance between corresponding particles (grey dot-dash line in Fig. 2):

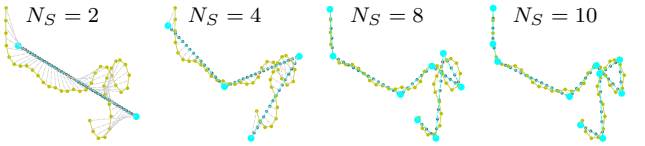

Fig. 2. Linear fitting of the simple model (blue) for a random goal configuration of the original model (yellow), for different N_S values. Grey dots represent sampled points in the simplified model, and grey dashed lines connect the corresponding points of both models.

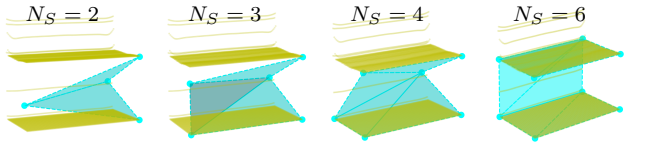

Fig. 3. Reduced mesh of the simple model (blue) for a side folded configuration of the original model (yellow), for different N_S values. The vertical dimension in these figures is scaled up for visualization purposes, to make the two halves of the folded cloth visually separate.

$$distance(\xi_G^S, \xi_G^O) = \sum_{i=1}^{N_E} \frac{\|q_i^O - q_i^S\|}{N_E} \tag{3}$$

where q_i^O represents the position of the i^{th} sampled point on the original model, and q_i^S represents the position of the i^{th} point sampled on the simplified model.

After QP minimization is complete, we use the same distance formulation above (Eq. 3) to implement the `Error` function in Alg. 1 (line 4) to compute the final distance between the two models.

An example piece-wise line fitting process for a random configuration of a rope is shown in Fig. 2. The first picture shows a simplified model with two particles and one line segment; the fit to the original model is quite poor. As the number of particles in the simplified model, N_S, is increased, we get better fits. Depending on the threshold set (Alg. 1, line 6), the simplification process can terminate, for example, at the eight-particle model or the ten-particle model.

If terminated, corresponding particles on the original model are extracted as graspable points, $\hat{V}_O$, and used to reduce the action space.

C. Mesh Simplification for 2-D Flat Models

For objects that can be approximated by surfaces, we use the Quadric Edge Collapse Decimation (QECD) method as the model simplification function of Alg. 1, which can simplify the model towards a given number of elements N_S. The basic element for mesh simplification is a triangle, which is composed of particles that can be shared between different triangles. QECD is a surface simplification algorithm based on the quadric error metrics proposed by Garland and Heckbert [25]. During simplification, pairs of vertices (particles) are contracted to one iteratively, until the target number of triangles, N_S, is achieved. We use the QECD implementation in the mesh processing library Meshlab [26].

We start from simplifying the original mesh to a model with $N_S = 2$ triangles (four particles, two of which are shared), and gradually increase the number of triangles until

the error is below the threshold, as in Alg. 1.

To implement the `Error` function and find the distance between the simplified mesh and the original one (Alg. 1, line 4), we use the Hausdorff distance [27], which is a widely used similarity metric for mesh and image comparison. The Hausdorff distance is defined as the maximum distance of a set to the nearest point in the other set, and in our case, is found by:

$$h(\xi_G^S, \xi_G^O) = \max_{q^S \in \xi_G^S} \{ \min_{q^O \in \xi_G^O} \left\| q^S - q^O \right\| \} \tag{4}$$

As shown in Fig. 3, a piece of side-folded cloth is initially approximated by a simple mesh with two triangles, which only covers half of each face. This is improved by adding more triangles to the simplified model. In the last picture, a simplified model with six triangles overlaps with the original model, giving us a satisfying approximation.

Finally, to find $\hat{V}_O$, the particles on the original model that is nearest to the particles on the simplified model are extracted as the preliminary particle set. Since creases on cloth (e.g. Fig. 5) can create many particles located extremely close to each other, we apply further filtering on this set based on geodesic distance, which reduces the redundancy in the set of extracted key particles.

D. Planning with Multiple Intermediate Goals

Some complex manipulation tasks may require picking up the object at points that cannot be identified from the final configuration only. For example, consider the task of tying a knot, where the rope should be straightened from a crumpled state before crossing the rope over itself (similar to the rope-folding goal shown in Fig. 4-(b)), and then getting one end of the rope through the resulting loop, and so on. Such plans, with intermediate goals, are often illustrated in knot or origami books. In our model-simplification-based action space reduction and motion planning framework, a sequence of goals can be processed and achieved one by one, which only requires repeating the process of extracting key particles and motion planning for each subgoal sequentially. However, in this paper, we do not consider the problem of computing the higher-level plan and producing the intermediate goals.

IV. EXPERIMENTS

We present three experiments on six different tasks. The first experiment aims to show that the geometric model simplification methods converge to a reasonable approximation of the original model at the goal configuration, which consequently gives a reduced set of graspable points. In the second experiment we quantify the improvement of the reduced action space on planning, in terms of planning iterations and quality of the computed solution (best value of the objective function per iteration). The third experiment validates the method on a real robot.

A. Planning algorithm

The planning problem defined in Eq. 2, can be solved by various off-the-shelf trajectory optimization or motion planning methods, such as CEM [28] or Rapidly-exploring Random Trees (RRT) [29]. For our experiments, we use CEM as an exemplary method for planning. CEM is a well-established, population-based, optimization algorithm, which has been applied to address plenty of manipulation problems including deformable objects. To find a solution, CEM repeatedly samples a set of trajectories from a multivariate Gaussian distribution, calculates the cost of each trajectory, identifies the elite individuals whose cost is below a given threshold, and uses these elite individuals to refit the Gaussian distribution for the next iteration.

B. Tasks

Our method is evaluated in six tasks, two for ropes and four for cloths. Among these tasks, Rope Straightening and Cloth Side Folding are borrowed directly from SoftGym [23], while remaining tasks are modified based on SoftGym.

1) Rope Straightening: Manipulating a rope from a randomized initial configuration to a straightened goal configuration (Fig. 4-(a)). Performance is measured by the error between the distance of the two endpoints and the original length of the rope.

2) Rope Folding: Manipulating a rope into a crossed triangle (Fig. 4-(b)), from an initially straightened state. Performance is measured on the bipartite matching distance between the final and goal positions of all particles.

3) Cloth Diagonal Folding: Folding a flattened cloth into half diagonally (Fid. 4-(c)). Performance is measured on the distance between the final and goal positions of all particles.

4) Cloth Side Folding: Folding the cloth into half sideways (Fig. 4-(d)).

5) Cloth Reflective Folding: Reflective folding is borrowed from origami folding where two consecutive folds are in opposite directions [30]. This task follows the goal configuration of the previous task (i.e. the side folding goal) as an intermediate goal. Then the final goal is to fold a quarter of the cloth back after (Fig. 4-(e)).

6) Cloth Underneath Folding: Underneath folding (e.g. Fig. 4-(f) and Fig. 5) refers to folding a quarter of the cloth under the rest of it, contrary to placing it on the top. This task follows on the goal configuration of the previous task (i.e. the cloth reflective folding goal) as an intermediate goal.

C. Simulation Experiment Results

Simulation experiments including geometric model simplification for action space reduction and CEM-based planning are conducted for the tasks defined above. All simulation tests are performed in SoftGym [23] on a workstation with Intel Xeon(R) CPU E5-2650 @2.60GHz, 32 GB of RAM, and Nvidia GeForce GTX 1070 GPU.

1) Model Simplification for Action Space Reduction: A summary of the model simplification for action space reduction results is shown in Fig. 4, including the approximation error curve, the simplified geometric model, and the reduced number of key particles extracted from the original geometric model at the goal configuration.

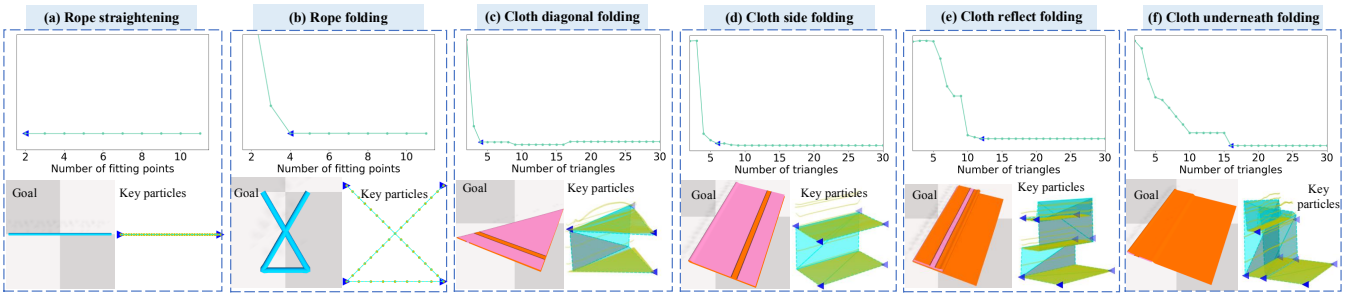

Fig. 4. Results of model simplification including approximation error history, image of the goal, and the simplified model with extracted key particles. The vertical dimension in the 'Key particles' figures are scaled up for visualization purposes, to make the folded cloth visually separate.

For the Rope Straightening task, as expected, only the two ends is enough to represent the goal state and reach the goal, as shown in Fig. 4-(a). For the Rope Folding task, as shown in Fig. 4-(b) a simplified model of four particles and three line segments are acquired, giving four corners on the original model as key particles.

As for cloth manipulation, our method finds much simpler geometric models than the original model for each task, which is illustrated in Fig. 4-(c-f). In Fig. 4-(c), the Hausdorff distance curve flattens at four triangles, which gives us a fine approximation to the original mesh with a much simplified model of four triangles for the diagonal folding task. Corresponding key particles are extracted and marked with blue triangles in the figure, which constitute the graspable points in the reduced action space. Similarly, for the rest of cloth folding tasks, a set of simplified geometric models with six, twelve, and sixteen triangles are acquired to approximate the original models and reduce the action space respectively.

Furthermore, the average model simplification and action space reduction time costs for the two rope manipulation tasks and four cloth manipulation tasks are 0.125 s and 0.730 s respectively, which are negligible compared to the time costs of motion planning.

2) Motion Planning: We compare CEM-based motion planning in the reduced action space (denoted as *CEM(Simple)*) with three baselines: CEM in the original action space (*CEM(Original)*), CEM in the randomly reduced action space (*CEM(Random)*), and a manually scripted policy, in which the picker grasps each key particle, moves above its target position and releases it. We run each CEM planner for 30 iterations. Each iteration of CEM takes 10.8 s for the rope, and 171.3 s for the cloth. The length of the action sequence T is 20 for rope tasks, and 15 for cloth tasks.

Since CEM is a stochastic method, we run *CEM(Simple)*, *CEM(Original)* and *CEM(Random)* methods ten times on each task. The planning cost per iteration and snapshots of the computed plans are shown in Fig. 7 (a-f). In the plots, the lines show the mean cost, while the line shadows show the 95% confidence interval over the ten runs.

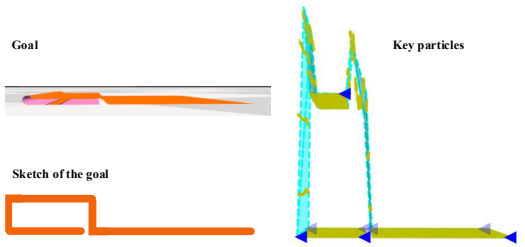

Fig. 5. Side view of the goal configuration of the underneath folding task and the simplified model (the height is scaled for visualization purposes) with extracted key particles (marked by blue triangles)

For rope straightening, *CEM(Simple)* finds a better plan than *CEM(Original)* using much fewer iterations, which is shown in Fig. 7-(a). The scripted policy finds a slightly worse solution because it keeps disrupting previously achieved states. For rope folding, *CEM(Simple)* still achieves a smaller cost. However, *CEM(Original)* also finds feasible plans with similar costs in about the same iterations. *CEM(Random)* finds the worst solution for both rope manipulation tasks, which reflects the significance of placing the key particles properly.

For the rope object, the size difference between the original action space, ($N_O = 40$), and the simplified action space ($\hat{N}_O = 2$ for the straightening task, and $\hat{N}_O = 4$ for the folding task), is not significant. Therefore we see modest or no gain between the two methods. For the cloth object however, the original model has $N_O = 10000$ particles, which is much larger than the number of extracted key particles $\hat{N}_O$, as shown in Sec. IV-C.1.

Therefore, from Fig. 7-(c-f), we can see that *CEM(Simple)* achieves better solution than *CEM(Original)* consistently in all four tasks of cloth manipulation. For instance, as shown in the planning cost curve for diagonal folding in Fig. 7-(c), *CEM(Simple)* converges to an optimal plan in only fifteen iterations, which successfully folds the cloth similar to the goal given in Fig. 4-(c). In contrast, *CEM(Original)* converges to a plan with a much higher cost which results in an incorrect final state. As the difficulty increases from side folding, reflective folding, to underneath folding tasks, the gap between the cost curve of *CEM(Simple)* and *CEM(Original)* also increases. As shown in Fig. 7(d-f), *CEM(Simple)* can find feasible plans to bring the cloth into the desired shape successfully, whereas *CEM(Original)* finds worse and worse results. The images of the manipulation process visualize the intermediate and final states. It can be seen that *CEM(Original)* does not achieve any cloth folding task using the maximum number of CEM iterations provided, while *CEM(Simple)* is able to converge to a solution. Besides, in all cloth folding tasks, *CEM(Random)* achieves slightly better results than *CEM(Original)*, which further demonstrates that not every particle on the original model

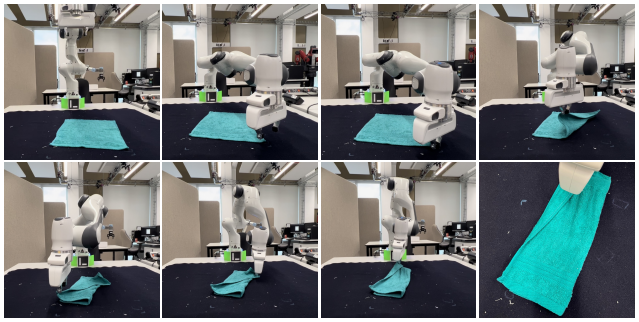

Fig. 6. Real robot demonstration of the cloth side folding task

is equally relevant to achieve the goal. Furthermore, in these more complex folding tasks, the advantage of the planner over the simple scripted behavior is increasingly evident.

D. Real Robot Experiment Results

To validate the effectiveness of the planned path in the real world, we used a Franka Panda robot to carry out the cloth diagonal folding and side folding tasks. The robot is mounted on a stationary table and a square cloth (30 cm $\times$ 30 cm) is placed in front of it at a pre-determined initial pose. For these demonstrations we did not use perception, and the robot executed the computed plan in open loop.

Firstly, the planned path is converted to several pick and place motions according to the defined action space. For picking a particular point on the cloth, Moveit [31] is used to plan a valid path to a certain height above the target using a position controller. After that an impedance controller is switched on to gradually lower the gripper until it contacts the cloth. The reason for using impedance control is to apply a certain amount of pressure before closing the gripper on the cloth.

Images of the real robot folding the cloth diagonally and sideways are shown in Fig. 1 and Fig. 6. Please refer to the video[1] to view the whole manipulation process. The Panda robot successfully achieves both tasks. Within each task, the gripper can re-grasp the cloth showing that our proposed controlling scheme for executing the plan is effective and the planned path in simulation can be executed successfully to fold the cloth. Nonetheless, the implementation of the real-world experiments is open-loop, and it suffers from the gap between the real cloth physics and the simulation.

V. CONCLUSION

In this paper, we proposed to reduce the action space for motion planning based on model simplification methods. Two workflows of model simplification and key particle extraction were developed for 1-D linear and 2-D flat objects respectively. Simulation and real robot experiments were conducted, which demonstrate that our proposed method can improve the efficiency and performance of a motion planner. The improvement of the reduced action space over the original one, in our results, increased with the number of particles in the original model, and the number of actions required to solve the task, suggesting that such model simplification

[1]https://youtu.be/nAYp42WV2g4

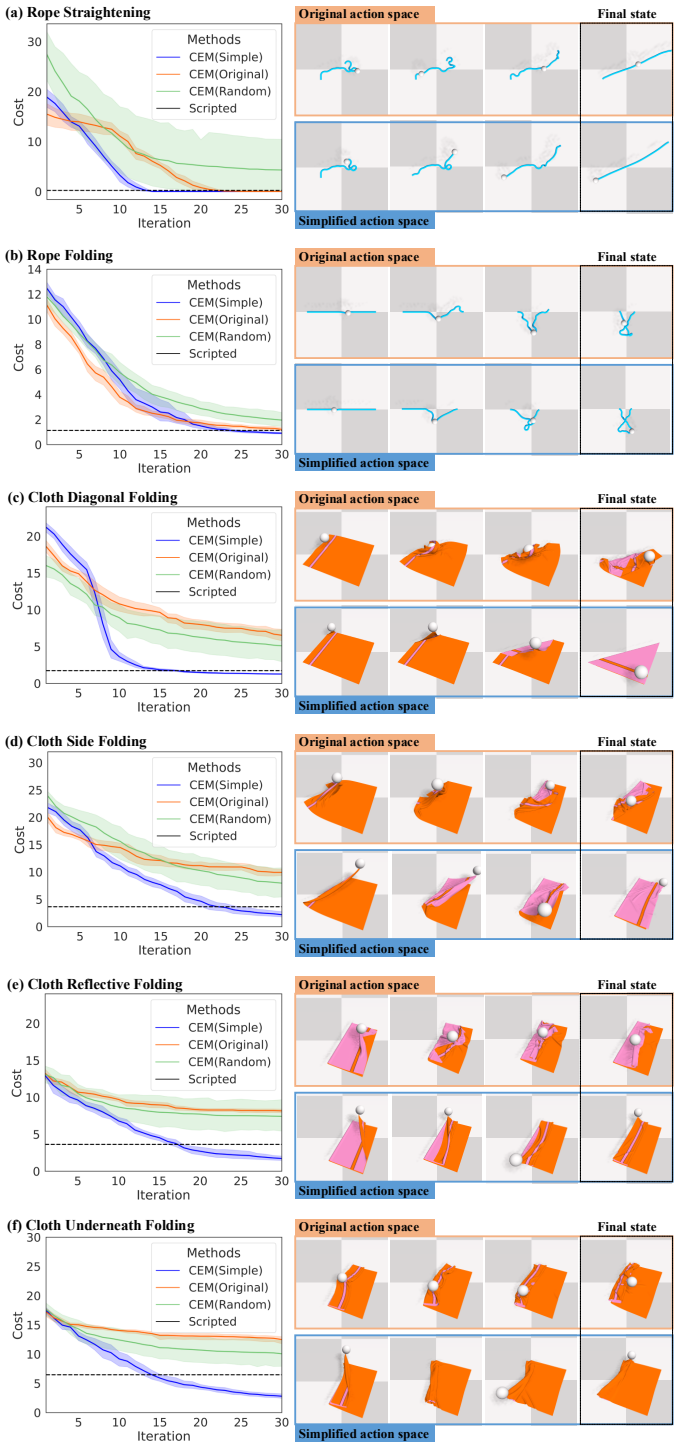

Fig. 7. Motion planning cost curve and the snapshots of manipulation plan found by *CEM(Original)* and *CEM(Simple)* for each task

and action space reduction may be critical for more complex tasks. Moreover, our proposed method is independent of the underlying dynamics model and the method used for motion planning. The computational bottleneck still exists due to time-consuming dynamics models and inefficient planning methods, which also suggests new directions for our future research. Besides, an exciting future line of investigation will be to combine the method proposed here with a high-level planner that generates intermediate goals for complex tasks.

References

[1] D. Seita, N. Jamali, M. Laskey, *et al.*, "Deep transfer learning of pick points on fabric for robot bed-making," in *The International Symposium of Robotics Research*, Springer, 2019, pp. 275–290.

[2] X. Li, X. Su, and Y.-H. Liu, "Vision-based robotic manipulation of flexible pcbs," *IEEE/ASME Transactions on Mechatronics*, vol. 23, no. 6, pp. 2739–2749, 2018.

[3] E. Torgerson and F. W. Paul, "Vision-guided robotic fabric manipulation for apparel manufacturing," *IEEE Control Systems Magazine*, vol. 8, no. 1, pp. 14–20, 1988.

[4] P. Jiménez, "Survey on model-based manipulation planning of deformable objects," *Robotics and computer-integrated manufacturing*, vol. 28, no. 2, pp. 154–163, 2012.

[5] A. Doumanoglou, A. Kargakos, T.-K. Kim, and S. Malassiotis, "Autonomous active recognition and unfolding of clothes using random decision forests and probabilistic planning," in *2014 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2014, pp. 987–993.

[6] Y. Li, Y. Yue, D. Xu, E. Grinspun, and P. K. Allen, "Folding deformable objects using predictive simulation and trajectory optimization," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2015, pp. 6000–6006.

[7] J. Zhu, A. Cherubini, C. Dune, *et al.*, "Challenges and outlook in robotic manipulation of deformable objects," *arXiv preprint arXiv:2105.01767*, 2021.

[8] H. Yin, A. Varava, and D. Kragic, "Modeling, learning, perception, and control methods for deformable object manipulation," *Science Robotics*, vol. 6, no. 54, eabd8803, 2021.

[9] S. Bhagat, H. Banerjee, Z. T. Ho Tse, and H. Ren, "Deep reinforcement learning for soft, flexible robots: Brief review with impending challenges," *Robotics*, vol. 8, no. 1, p. 4, 2019.

[10] V. E. Arriola-Rios, P. Guler, F. Ficuciello, D. Kragic, B. Siciliano, and J. L. Wyatt, "Modeling of deformable objects for robotic manipulation: A tutorial and review," *Frontiers in Robotics and AI*, vol. 7, p. 82, 2020.

[11] L. Sun, G. Aragon-Camarasa, P. Cockshott, S. Rogers, and J. P. Siebert, "A heuristic-based approach for flattening wrinkled clothes," in *Conference Towards Autonomous Robotic Systems*, Springer, 2013, pp. 148–160.

[12] J. Maitin-Shepard, M. Cusumano-Towner, J. Lei, and P. Abbeel, "Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding," in *2010 IEEE International Conference on Robotics and Automation*, IEEE, 2010, pp. 2308–2315.

[13] S. Miller, J. Van Den Berg, M. Fritz, T. Darrell, K. Goldberg, and P. Abbeel, "A geometric approach to robotic laundry folding," *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 249–267, 2012.

[14] J. Matas, S. James, and A. J. Davison, "Sim-to-real reinforcement learning for deformable object manipulation," in *Conference on Robot Learning*, PMLR, 2018, pp. 734–743.

[15] W. Yan, A. Vangipuram, P. Abbeel, and L. Pinto, "Learning predictive representations for deformable objects using contrastive estimation," *arXiv preprint arXiv:2003.05436*, 2020.

[16] S. Arnold, D. Tanaka, and K. Yamazaki, "Cloth manipulation planning on basis of mesh representations with incomplete domain knowledge and voxel-to-mesh estimation," *arXiv preprint arXiv:2103.08137*, 2021.

[17] Z. Huang, X. Lin, and D. Held, "Mesh-based dynamics with occlusion reasoning for cloth manipulation," *arXiv preprint arXiv:2206.02881*, 2022.

[18] X. Ma, D. Hsu, and W. S. Lee, "Learning latent graph dynamics for deformable object manipulation," *arXiv preprint arXiv:2104.12149*, 2021.

[19] R. Hoque, D. Seita, A. Balakrishna, *et al.*, "Visuospatial foresight for multi-step, multi-task fabric manipulation," *arXiv preprint arXiv:2003.09044*, 2020.

[20] X. Lin, Y. Wang, Z. Huang, and D. Held, "Learning visible connectivity dynamics for cloth smoothing," in *Conference on Robot Learning*, PMLR, 2022, pp. 256–266.

[21] Z. Xu, C. Chi, B. Burchfiel, E. Cousineau, S. Feng, and S. Song, "Dextairity: Deformable manipulation can be a breeze," *arXiv preprint arXiv:2203.01197*, 2022.

[22] T. Power and D. Berenson, "Keep it simple: Data-efficient learning for controlling complex systems with simple models," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1184–1191, 2021.

[23] X. Lin, Y. Wang, J. Olkin, and D. Held, "Softgym: Benchmarking deep reinforcement learning for deformable object manipulation," *arXiv preprint arXiv:2011.07215*, 2020.

[24] Y. Li, J. Wu, R. Tedrake, J. B. Tenenbaum, and A. Torralba, "Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids," *arXiv preprint arXiv:1810.01566*, 2018.

[25] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 1997, pp. 209–216.

[26] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, G. Ranzuglia, *et al.*, "Meshlab: An open-source mesh processing tool.," in *Eurographics Italian chapter conference*, Salerno, Italy, vol. 2008, 2008, pp. 129–136.

[27] N. Aspert, D. Santa-Cruz, and T. Ebrahimi, "Mesh: Measuring errors between surfaces using the hausdorff

distance," in *Proceedings. IEEE international conference on multimedia and expo*, IEEE, vol. 1, 2002, pp. 705–708.

[28] R. Rubinstein, "The cross-entropy method for combinatorial and continuous optimization," *Methodology and computing in applied probability*, vol. 1, no. 2, pp. 127–190, 1999.

[29] I. Noreen, A. Khan, and Z. Habib, "Optimal path planning using rrt* based approaches: A survey and future directions," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 11, 2016.

[30] D. J. Balkcom and M. T. Mason, "Robotic origami folding," *The International Journal of Robotics Research*, vol. 27, no. 5, pp. 613–627, 2008.

[31] S. Chitta, "Moveit!: An introduction," in *Robot Operating System (ROS)*, Springer, 2016, pp. 3–27.