FabricFlowNet: Bimanual Cloth Manipulation with a Flow-based Policy

Thomas Weng, Sujay Bajracharya, Yufei Wang, Khush Agrawal, and David Held
Robotics Institute, Carnegie Mellon University, USA
{tweng, sbajrach, yufeiw2, khusha, dheld}@andrew.cmu.edu

Abstract: We address the problem of goal-directed cloth manipulation, a challenging task due to the deformability of cloth. Our insight is that optical flow, a technique normally used for motion estimation in video, can also provide an effective representation for corresponding cloth poses across observation and goal images. We introduce FabricFlowNet (FFN), a cloth manipulation policy that leverages flow as both an input and as an action representation to improve performance. FabricFlowNet also elegantly switches between dual-arm and single-arm actions based on the desired goal. We show that FabricFlowNet significantly outperforms state-of-the-art model-free and model-based cloth manipulation policies. We also present real-world experiments on a bimanual system, demonstrating effective sim-to-real transfer. Finally, we show that our method generalizes when trained on a single square cloth to other cloth shapes, such as T-shirts and rectangular cloths. Video and other supplementary materials are available at: https://sites.google.com/view/fabricflownet.

Keywords: deformable object manipulation, optical flow, bimanual manipulation

1 Introduction

Cloth manipulation has a wide range of applications in domestic and industrial settings. However, it has posed a challenge for robot manipulation: compared to rigid objects, fabrics have a higher-dimensional configuration space, can be partially observable due to self-occlusions in crumpled configurations, and do not transform rigidly when manipulated. Early approaches for cloth manipulation relied on scripted actions; these policies are typically slow and do not generalize to arbitrary cloth goal configurations [1, 2, 3].

Recently, learning-based approaches have been explored for cloth manipulation [4, 5, 6, 7, 8], including model-free reinforcement learning to obtain a policy [9, 10]. For a cloth manipulation policy to be general to many different objectives, it must receive a representation of the current folding objective. A standard approach for representing a goal-conditioned policy is to input an image of the current cloth configuration together with an image of the goal [9, 8].

We will show a number of downsides to such an approach when applied to cloth manipulation. First, the policy must learn to reason about the relationship between the current observation and the goal, while also reasoning about the action needed to obtain that goal. These are both difficult learning problems; requiring the network to reason about them jointly exacerbates the difficulty. Additionally, previous work has used reinforcement learning (RL) to try to learn such a policy [9, 10]; however, a reward function is a fairly weak supervisory signal, which makes it difficult to learn a complex cloth manipulation policy. Finally, while many desirable folding actions are more easily and accurately manipulated with bimanual actions, previous learning-based methods for goal-conditioned cloth manipulation have been restricted to single-arm policies.

In this paper, we introduce FabricFlowNet (FFN), a goal-conditioned policy for bimanual cloth manipulation that uses optical flow to improve policy performance (see Fig. 1). Optical flow has typically been used for video-related tasks such as object tracking and estimating camera motion. We demonstrate that flow can also be used in the context of policy learning for cloth manipulation; we use an optical flow-type network to estimate the relationship between the current observation and a sub-goal. We use flow in two ways: first, as an input representation to our policy; second, after estimating the pick points for a pick-and-place policy, we query the flow image to determine the place

5th Conference on Robot Learning (CoRL 2021), London, UK.

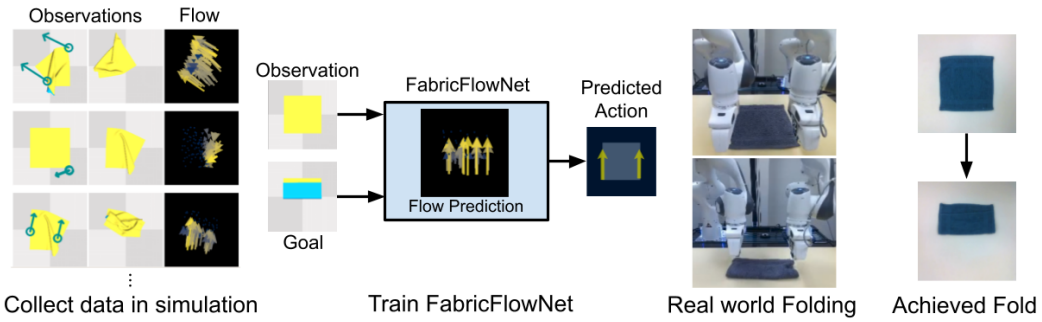


Figure 1: FabricFlowNet (FFN) overview. We collect a dataset of random actions and ground truth flow to train FFN. FFN learns to predict flow and uses it as both an input and action representation in a manipulation policy. FFN successfully performs single and dual-arm folding in the real world.

actions. Our method is learned entirely with supervised learning, leveraging ground truth particles from simulation. Our method learns purely from random actions without any expert demonstrations during training and without reinforcement learning.

Our learned policy can perform bimanual manipulation and switches easily between dual and single-arm actions, depending on what is most suitable for the desired goal. Our approach significantly outperforms our best efforts to extend recent single-arm cloth manipulation approaches to bimanual manipulation tasks [4, 9]. We present experiments on a dual-arm robot system and in simulation evaluating our method's cloth manipulation performance. FabricFlowNet outperforms state-of-the-art model-based and model-free baselines, and we provide extensive ablation experiments to demonstrate the importance of each component of our method to the achieved performance. Our method also generalizes with no additional training to other cloth shapes and colors. This paper contributes:

- A novel flow-based approach for learning goal-conditioned cloth manipulation policies that can perform dual-arm and single-arm actions.
- A test suite for benchmarking goal-conditioned cloth folding algorithms encompassing and expanding on goals used in previous literature [4, 9, 11]; we perform extensive experiments using this test suite to evaluate FabricFlowNet (FFN), baselines [4, 9], and ablations, demonstrating that FFN outperforms previous approaches.
- Experiments to demonstrate that FFN generalizes to other cloth colors and shapes, even without training on such variations.

2 Related Work

Bimanual Manipulation. A large body of research exists on dual-arm, or bimanual, manipulation [12]. Dual-arm systems allow for more complex behaviors than single-arm systems at the cost of greater planning complexity [13, 14], leading to research on closed kinematic chain planning [15, 16], composable skill learning [17, 18], and rewarding synergistic behavior [19]. Prior work has also explored bimanual cloth manipulation [20], including establishing a diverse set of benchmark tasks [21]. Cloth manipulation is a highly underactuated task, and bimanual manipulation enables controlling multiple cloth points [22]. A common approach for cloth flattening is to lift a cloth with one arm and regrasp it with the other arm until it reaches the flattened configuration [23, 24, 1, 2, 3]. Previous work in this direction uses hard-coded policies [1, 2, 3], whereas we learn to achieve arbitrary folded configurations. Tanaka *et al.* [25] learn bimanual actions for goal-conditioned folding, using a voxel-based dynamics model to predict how actions will change the cloth state. However, optimizing this dynamics model can slow down inference time compared to our model-free approach. Dynamic bimanual manipulation has also been explored in simulation from ground-truth keypoints [26] and for unfolding cloth in the real world [27]; we perform real-world bimanual folding using depth image observations.

Learning for Cloth Manipulation. Prior works have proposed various hand-defined representations for cloth manipulation, such as parameterized shape models [28] or binary occupancy features [29]. Recent approaches use contrastive learning to learn pixel-wise latent embeddings for

cloth [11, 30]. Both contrastive learning [11] and goal-conditioned transporter networks [8] have been applied to imitate expert demonstrations. Our approach doesn't require expert actions, just sub-goal states provided at test-time to define the task. In contrast to these previous representations, our method uses a flow-based representation, which we found to significantly outperform previous methods for goal-based cloth manipulation.

Other approaches have applied policy learning techniques to single-arm cloth smoothing [10, 5]. In contrast, we learn a policy that performs either single and dual-arm cloth manipulation; further, our focus is on goal-conditioned cloth folding, rather than smoothing. For cloth manipulation, Lee *et al.* [9] learns a model-free value function, but is limited by its discrete action space, and further, they do not use a flow-based representation, which we show leads to large benefits. Prior methods for learning goal-conditioned policies have used self-supervised learning to learn an inverse dynamics model for rope [7, 31] but such approaches have not been demonstrated for cloth manipulation. Lippi *et al.* [32] plan cloth folding actions in latent space, but do not demonstrate generalization to unseen cloth shapes. Other papers use an online simulator [23], or learn a cloth dynamics model in latent space [6], pixel-space [4], or over a graph of keypoints [33]. Unlike these model-based methods, our method is model-free and does not require online simulation or time-expensive CEM planning, leading to much faster inference. Further, we compare our approach to state-of-the-art approaches for cloth manipulation [9, 4] and show significantly improved performance.

Optical Flow for Policy Learning. Optical flow is the task of estimating per-pixel correspondences between two images, typically for video-related tasks such as object tracking and motion estimation. State-of-the-art approaches use convolutional neural networks (CNN) to estimate flow [34, 35, 36]. Optical flow between successive observations has previously been used as an input representation to capture object motion for peg insertion [37] or dynamic tasks [38]. Within the domain of cloth manipulation, Yamazaki *et al.* [39] similarly use optical flow on successive observations to identify failed actions. We use flow not to represent motion between successive images, but to correspond the cloth pose between observation and goal images, and to determine the placing action for folding. Argus *et al.* [40] use flow in a visual servoing task to compute residual transformations between images from a demonstration trajectory and observed images. In contrast, we learn a policy with flow to determine what cloth folding actions to take, not how to servo to a desired pose.

3 Learning a Goal-Conditioned Policy for Bimanual Cloth Manipulation

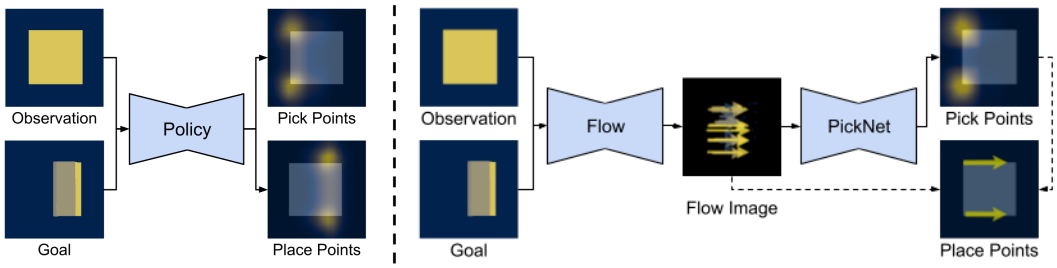
3.1 Problem Definition

Our objective is to enable a robot to perform cloth folding manipulation tasks. Let each task be defined by a sequence of sub-goal observations $\mathcal{G} : \{x_1^g, x_2^g, \ldots, x_N^g\}$, each of which can be achieved by a single (possibly bimanual) pick-and-place action from the previous sub-goal. We require sub-goals, rather than a single goal, because a folded cloth can be highly self-occluded such that a single goal observation fails to describe the full goal state. Defining a task using a sequence of sub-goals is found in other recent work [31]. Similar to prior work [31, 7], even if the sub-goals are obtained from an expert demonstration, we nonetheless do not assume access to the expert actions; this is a realistic assumption if the sub-goals are obtained from visual observations of a human demonstrator.

We assume that the agent does not have access to the sub-goal sequence $\mathcal{G}$ during training that it must execute during inference. Thus, the agent must learn a general goal-conditioned policy $a_t = \pi(x_t, \mathcal{G})$, where x_t is the current observation of the cloth and $a_t \in \mathcal{A}$ is the action selected by the policy. In our approach, we input each sub-goal x_i^g sequentially to our policy: $a_t = \pi(x_t, x_i^g)$. A goal recognizer [31] can also be used to decide which sub-goal observation to input at each timestep. For convenience, we will interchangeably refer to x_i^g as a goal or sub-goal.

3.2 Overview

A common approach for a goal-conditioned policy is to input the current observation x_t and the goal observation x_i^g directly into to a neural network representation of a policy [31, 9] or a Q-function [10, 9]. However, the network must reason simultaneously about the relationship between the observation and the goal, as well as the correct action to achieve that goal. Our first insight is that we can improve performance by separating these two components: we will learn to reason about the relationship between the observation and the goal, and separately use this relationship to reason



(a) Naive system (b) FabricFlowNet (FFN)

Figure 2: (a) A naive approach to goal-conditioned policy learning is to input observation and goal images directly to the policy and predict the action. (b) FabricFlowNet separates representation learning from policy learning; it first estimates the correspondence between the observation and goal as a flow image. The flow is then used as the input to PickNet for pick point prediction, and as a way to compute place points without requiring additional learning.

over actions. Specifically, we represent this relationship using a "flow image" f, which indicates the correspondence between the current observation x_t and sub-goal x_i^g. Thus we propose using the flow image f as an improved input representation of the policy, rather than directly inputting the observation x_t and goal observation x_i^g.

Our second insight is that we can also use flow in the output representation of the policy. We use a pick and place action space; prior methods that learn pick and place policies for deformable object manipulation predict place points using the policy network, either explicitly [5, 8, 6, 7, 31, 10] or implicitly by transforming the inputs to a Q-function [9]. Instead, we simplify the problem by leveraging flow: our policy network only learns to predict the *pick* points. For the place point, we query the flow image f for the flow vector starting at the predicted pick location, and use the endpoint of that vector as the place point.

We demonstrate that using flow in the two ways described above for our policy achieves significantly improved performance compared to prior work. Furthermore, our approach extends naturally to dual-arm manipulation, allowing us to easily transition between single and dual-arm actions.

A schematic overview of our system can be found in Fig. 2b. We first compute the flow f between the current observation x_t and goal x_i^g. Next, we input the flow f to a policy network (PickNet), which outputs pick points p_i. We then query the flow image $f(p_i)$ to determine the place points for each robot arm. Further details of our approach are described below.

3.3 Estimating Flow between Observation and Goal Images

We learn flow to use it as an input representation to our pick prediction network, and as an action representation for computing place points. Given an observed depth image x_t and desired goal depth image x_i^g, we estimate the flow $f = (f^1, f^2)$, mapping each pixel (u, v) in x_t to its corresponding coordinates $(u', v') = (u + f^1(u), v + f^2(v))$ in x_i^g. This task formulation differs from standard optical flow tasks as the input image pairs (x_t, x_i^g) are not consecutive images from video frames.

To capture the complex correspondences between x_t and x_i^g, we train a convolutional neural network to estimate the flow image f (see Appendix for details). The training loss we use to supervise the network is endpoint error (EPE), the standard error for optical flow estimation. EPE is the Euclidean distance between the predicted flow vectors f and the ground truth f^*, averaged over all pixels: $\mathcal{L}_{\text{EPE}} = \frac{1}{N}\sum_{i=1}^{N} \|f^* - f\|_2$. We use a cloth simulation to collect training examples with ground truth flow. The simulator provides the ground-truth correspondence between the particles of the cloth in different poses. The simulation cloth particles are not as dense as the depth image pixels; as a result, we only have ground-truth flow supervision for a sparse subset of the pixels that align with the cloth particles. Thus, we mask the loss to only supervise the flow for the pixels that align with the location of the cloth particles. We train the flow network using data collected from random actions. See Sec. 3.6 for more details on the simulator, data collection, and network training.

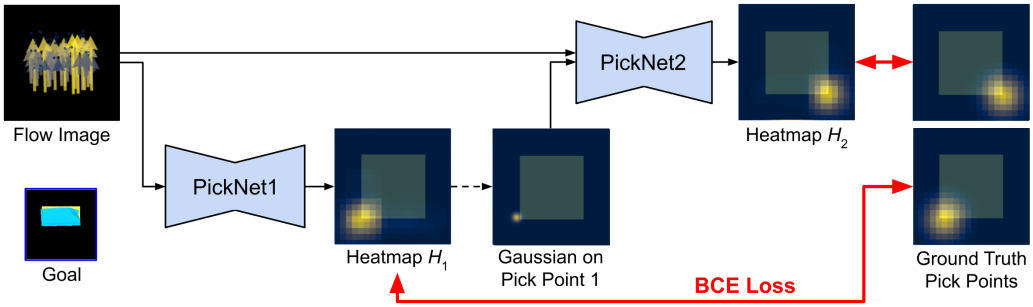


Figure 3: PickNet architecture. We utilize a two-network architecture for bimanual manipulation, where the second pick point is conditioned on the prediction of the first pick point.

3.4 Learning to Predict Pick Points

Our bimanual action space $\mathcal{A}$ consists of actions $a = (p_1, p_2, q_1, q_2)$, where p and q are the pick and place points respectively, paired according to the subscripts. We train a neural network called PickNet to estimate the pick points p_1, p_2. Crucially, the input to PickNet is a flow image f, estimated between the current depth image x_t and the desired goal depth image x_i^g, as described in the previous section. The flow image indicates, for each pixel (u, v) in the current observation, the location $f(u, v)$ that the pixel has moved to in the goal observation. Our flow network (Sec. 3.3 above) reasons about the observation-goal relationship, so that the policy network (PickNet) only needs to reason about the action, specifically the two pick points (p_1, p_2); computing the place points is described in Sec. 3.5.

For dual-arm actions, the pick points must be estimated conditionally, as the location of pick point p_1 on the cloth influences the optimal location of pick point p_2, and vice versa. To decouple this conditional estimation problem, we propose a two-network architecture, PickNet1 and PickNet2, to estimate the pick points (see Fig. 3). This architecture was inspired by Wu *et al.* [10], which used two networks for pick-conditioned placing; we instead use two networks to condition dual-arm picking. PickNet1 is a fully convolutional network that receives flow image f as input and outputs a single heatmap H_1 estimating the optimal pick points for arm 1. We compute the first pick point as $p_1 = \arg\max_p H_1(p)$. The second network, PickNet2, predicts the second arm's pick point p_2 conditioned on p_1; PickNet2 takes as input both the flow image f and an additional image with a 2D Gaussian centered on p_1, and is otherwise identical to PickNet1. PickNet2 outputs heatmap H_2, from which we compute the second pick point: $p_2 = \arg\max_p H_2(p)$. The two-network architecture decouples the conditionally dependent pick point predictions and does not require us to resort to heuristics to extract two pick points from a single heatmap. We refer to PickNet1 and PickNet2 together as "PickNet."

To train PickNet, we collect a dataset of random actions (see Sec. 3.6 for details) and record the current observation x_t, the bimanual action $a = (p_1, p_2, q_1, q_2)$, and the next observation x_{t+1}. We also estimate the flow f from x_t to x_{t+1}, as explained in Sec. 3.3. We create ground truth pick heatmaps H_i^* for arm i using the recorded random action a, by placing a 2D Gaussian $\mathcal{N}(p_i, \sigma)$ on each ground truth pick location p_i. We then supervise PickNet using the binary cross-entropy (BCE) loss between predicted heatmaps H_1, H_2 and ground truth heatmaps H_1^*, H_2^*. However, it might be unclear to the network which pick point should be output by PickNet1 and which should be output by PickNet2. We compute the loss for both possible correspondences and use the minimum:

$$\begin{aligned} l_{\text{BCE}}(H_i, H_j, H_i^*, H_j^*) &= \text{BCE}(H_i, H_i^*) + \text{BCE}(H_j, H_j^*) \\ \mathcal{L}_{\text{Pick}} &= \min[l_{\text{BCE}}(H_1, H_2, H_1^*, H_2^*), l_{\text{BCE}}(H_2, H_1, H_1^*, H_2^*)] \end{aligned} \tag{1}$$

At inference time, PickNet outputs the pick points p_1, p_2, computed from the argmax of H_1, H_2 respectively, as described above.

3.5 Estimating the Place Points from Flow

After estimating the pick points p_1, p_2 from flow, the remaining step to predict a bimanual pick and place action $a = (p_1, p_2, q_1, q_2)$ is to estimate the place points q_1, q_2. A straightforward approach would be to train the network to predict place points q_1, q_2, similar to the pick points p_1, p_2 as

described above (see Fig. 2a). Instead, our approach uses the flow image to find the place points, so that the place points do not have to be learned separately.

Our approach makes the assumption that, to achieve a desired subgoal configuration, the point picked on the cloth should be moved to its corresponding position in the goal image (which is estimated by the flow). This is a simplifying assumption, since it is possible that the picked point will shift slightly after it is released by the gripper; our method does not take into account such small movements. Using this assumption, to compute the place points q_1, q_2, we query the flow f at each pick point p_1, p_2 to estimate the delta between the pick point location in the observation image and the corresponding location of the pick points in the goal image. We use these predicted correspondences as the place points: $q_i = f(p_i) + p_i$, for each arm i.

Action predictions estimated by our approach can produce nearly overlapping pick and place points, indicating that arm 1 and arm 2 should perform identical actions. We observe this behavior from PickNet when the goal is best achieved with a single-arm action, rather than a bimanual one. On a real robot, grippers are likely to collide if grasping points that are too close. Therefore, to switch between executing a single-arm or bimanual action, we compute the L2 pixel distance between pick points $d_{\text{pick}} = \|p_1 - p_2\|_2$ and place points $d_{\text{place}} = \|q_1 - q_2\|_2$. We use a single-arm action when either distance is smaller than a threshold α, which we set to 30 for all experiments.

3.6 Implementation Details

We use SoftGym [41], an environment for cloth manipulation built on the particle-based simulator Nvidia Flex, to collect training datasets. The simulator models cloth as particles connected by springs. We use pickers that simulate a grasping action by binding to the nearest cloth particle within a threshold to execute pick and place actions in SoftGym. We collect data by taking random actions, biased towards grasping corners of the cloth. We demonstrate that we are able to train our method in SoftGym and then transfer the policy to the real world. Details on the data collection, as well as the network architecture and training details, can be found in Appendix Sec. A.1.

4 Experiments

4.1 Simulation Experiments

Experiment Setup. We evaluate FabricFlowNet (FFN) and compare to state-of-the-art baselines in the SoftGym [41] simulator; real-world evaluations are below in Sec. 4.2. Our experiments focus on folding tasks, and we assume that a cloth smoothing method (*e.g.,* [27, 5]) is used to flatten the cloth before folding is executed. Our error metric is the average particle position error between the achieved and goal cloth configuration. We evaluate on two sets of goals: 40 *one-step* goals that can be achieved with a single fold action, and 6 *multi-step* goals that require multiple folding actions. The multi-step goals each consist of a sequence of sub-goal images, with the next sub-goal presented after each action. This protocol follows from our problem formulation in Sec. 3.1, and is similar to the protocol in Nair *et al.* [7]. Our goals include test goals from Ganapathi *et al.* [11] and Lee *et al.* [9] that are achievable with one arm, as well as additional goals more suitable for two-arm actions (see Appendix Fig. S2 for the full set of goals).

We compare our method to Fabric-VSF [4], which learns a visual dynamics model and uses CEM to plan using the model. We only use Fabric-VSF with RGB-D input, as depth-only input performs poorly for folding tasks [4]. FabricFlowNet only uses depth and does not rely on RGB, which enables our method to transfer easily to the real world without extensive domain randomization. We also compare to Lee *et al.* [9], a model-free approach. We extend the the original single-arm method to a dual-arm variant and compare against both. For both our method and the baselines, we only allow each method to perform one pick-and-place action for each subgoal (e.g. one pick and place action for each single-step goals). Additional baseline details can be found in the Appendix.

4.1.1 Simulation Results

Table 1 contains our simulation results for all methods. We report average particle distance error (in mm) for one-step goals only, multi-step goals only, and over both one-step and multi-step goals. Our results show that FFN achieves the lowest error over all goals and has the fastest inference time.

Table 1: Mean Particle Distance Error (mm) and Inference Time (sec) on Cloth Folding Goals

Method	One Step (n=40)	Multi Step (n=6)	All (n=46)	Inference Time
Lee *et al.*, 1-Arm [9]	16.18 ± 08.38	26.20 ± 16.31	17.49 ± 10.10	~ 0.04
Lee *et al.*, 2-Arm	36.62 ± 14.51	47.71 ± 21.95	38.07 ± 15.82	~ 0.04
Fabric-VSF [4]	6.31 ± 06.55	$\mathbf{21.33 \pm 11.20}$	8.27 ± 08.90	~ 420
FabricFlowNet (Ours)	$\mathbf{4.46 \pm 02.62}$	25.04 ± 22.88	$\mathbf{7.14 \pm 11.06}$	$\sim \mathbf{0.007}$

We also investigate whether using flow as a goal recognizer improves performance. When an observation closely matches the goal, the flow for all points is close to zero. We leverage this fact by evaluating FFN with "iterative refinement": we allow the policy to take multiple actions per subgoal to try to further minimize the flow between the observation and subgoal. When the average flow between observation and current subgoal reaches a minimum threshold, the policy moves forward to the next subgoal. FFN with iterative refinement achieves a mean error of 6.62 over all goals vs. 7.14 without refinement. Additional details on iterative refinement can be found in the Appendix, along with additional results from baseline variants, crumpled initial configurations, and end-to-end training.

4.1.2 Ablations

We run series of ablations to evaluate the importance of the components of our system; results averaged over all 46 goals are in Table 2. Additional details and results are in Appendix Sec. D. Our ablations are designed to answer the following questions:

What is the benefit of using flow as input? We modify PickNet to receive depth images of the observation and goal as input to the network ("NoFlowIn"), as is commonly done in previous work on goal-conditioned RL [9, 8]. In this ablation, the PickNet needs to reason about both the relationship between the observation and the goal, as well as the action. In contrast, our method uses the flow network to compare the observation and goal; the picknet separately reasons about the action. **What is the benefit of using flow to choose the place point?** In this ablation, we train a network to predict the place points directly ("NoFlowPlace"). This is in contrast to our approach where we use the flow field, evaluated at the pick point $f(p_i)$, to compute the place point q_i for arm i. Our approach leads to a 32.4% improvement, showing the benefit to using flow as an action representation.
What is the performance with no flow? We combine the above two ablations and remove flow entirely, ("NoFlow"; ours has 60.4% improvement). The above ablations all indicate the strong benefit of using flow as both an input and action representation for cloth manipulation.
What is the benefit of biasing the data collection to grasp corners? Our method uses prior knowledge about cloth folding tasks to bias the training data and pick at corners of the cloth. In this ablation, we choose pick points randomly ("NoCornerBias", ours has 35.5% better performance).
What is the performance with a simpler architecture? We also compare our architecture for PickNet (Sec. 3.4) to a simpler architecture that takes as input the flow image I_f and outputs a two heatmaps, one for each pick point ("NoSplitPickNet"; ours has 2.1% better performance).
Does the loss formulation in Eq. 1 improve performance? We compare our method to an ablation where the first ground-truth heatmap is used to supervise PickNet1 and similarly for the second, i.e. $\mathcal{L}_{\text{Pick}} = l_{\text{BCE}}(H_1, H_2, H_1^*, H_2^*)$. ("NoMinLoss"; ours has similar performance).

Table 2: Mean Particle Distance Error (mm) for Ablations over All Goals (n=46)

NoFlowIn	NoFlowPlace	NoFlow	NoCornerBias	NoSplitPickNet	NoMinLoss	FFN (Ours)
9.37	10.56	18.02	11.07	7.29	7.15	$\mathbf{7.14}$

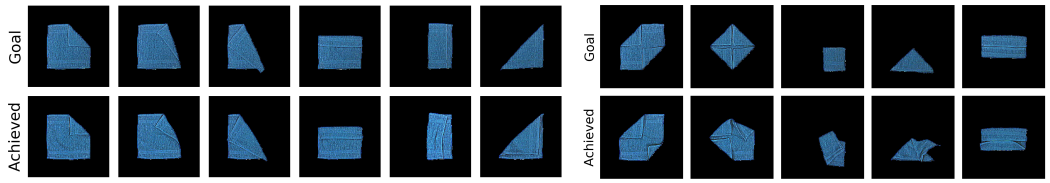
4.2 Real World Experiments

We evaluate FabricFlowNet in the real world and demonstrate that our approach successfully manipulates cloth on a real robot system.

Experiment Setup. Our robot system consists of two 7-DOF Franka Emika Panda arms and a single wrist-mounted Intel RealSense D435 sensor (See Fig. 1). We plan pick and place trajectories using MoveIt! [42]. We evaluate on a 30x30 cm towel, using 6 single-step and 5 multi-step goals (see Fig. 4) that form a representative subset of our simulation test goals.

To transfer from simulation to the real world, we align the depth between real and simulated images by subtracting the difference between the average depth of the real support surface (i.e. the table) and the simulated surface. We mask the cloth by color-thresholding the background; see Appendix for details. We found that these simple techniques were sufficient to transfer the method trained entirely in simulation to the real world, because we use only depth images as input. Simulated depth images match reasonably well to real depth images, unlike RGB images.

4.2.1 Real World Results

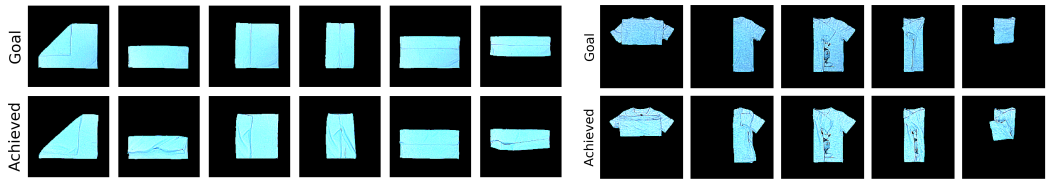


(a) One-step Square Cloth (b) Multi-step Square Cloth

Figure 4: Qualitative results for FFN on real world experiments. FFN only takes depth images as input, allowing it to easily transfer to cloth of different colors.

Fig. 4 provides qualitative real world results, showing that we successfully achieve many of the goals. Our website (link in abstract) contains videos of these trials.

We compare FabricFlowNet to the NoFlow ablation from Sec. 4.1.2. Both methods used the same sim-to-real techniques described in the previous section. While we do not have access to the true cloth position error in the real world, Intersection-over-Union (IoU) on the achieved cloth masks serves as a reasonable proxy metric [9]. FFN achieves 0.80 mean IoU over 3 trials for the square cloth, compared to 0.53 for NoFlow. See the Appendix for additional details.



(a) Rectangular Cloth (b) Printed T-shirt

Figure 5: Generalization to new cloth shapes for FFN trained only on a square cloth in simulation. FFN achieves single and multi-step goals for rectangular fabric and a printed T-shirt.

Generalization. In addition to evaluating the folding policy on square cloth for various goal configurations, we also test the generalization of our method to other shapes of cloth. We evaluate the performance of FFN trained only on a square cloth on folding goals for a rectangular cloth as well as a T-shirt. These fabrics are also thinner than the square blue towel used in the real world experiments above. Fig. 5 shows that FFN trained on a square yellow cloth in simulation is able to generalize to other cloth shapes, textures, and colors (FFN only receives depth images as input). See Appendix for additional details.

5 Conclusion

In this work we present FabricFlowNet, a method which utilizes flow to learn goal-conditioned fabric folding. We leverage flow to represent the correspondence between observations and goals, and as an action representation. The method is trained entirely using random data in simulation. Our results show that separating the correspondence learning and the policy learning can improve performance on an extensive suite of single- and dual-arm folding goals in simulated and real environments. Our experiments also demonstrate generalization to different fabric shapes, textures, and colors. Future work on flow-based fabric manipulation could incorporate actions beyond pick and place, such as parameterized trajectories or dynamic actions.

Acknowledgments

This work was supported by the National Science Foundation (NSF) Smart and Autonomous Systems Program (IIS-1849154), LG Electronics, and a NSF Graduate Research Fellowship (DGE-1745016).

References

[1] 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*, pages 2308–2315, 2010. doi: 10.1109/ROBOT.2010.5509439.

[2] A. Doumanoglou, J. Stria, G. Peleka, I. Mariolis, V. Petrík, A. Kargakos, L. Wagner, V. Hlaváč, T.-K. Kim, and S. Malassiotis. Folding clothes autonomously: A complete pipeline. *IEEE Transactions on Robotics*, 32(6):1461–1478, 2016. doi:10.1109/TRO.2016.2602376.

[3] C. Bersch, B. Pitzer, and S. Kammel. Bimanual robotic cloth manipulation for laundry folding. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1413–1419, 2011. doi:10.1109/IROS.2011.6095109.

[4] R. Hoque, D. Seita, A. Balakrishna, A. Ganapathi, A. Tanwani, N. Jamali, K. Yamane, S. Iba, and K. Goldberg. VisuoSpatial Foresight for Multi-Step, Multi-Task Fabric Manipulation. In *Proceedings of Robotics: Science and Systems*, Corvalis, Oregon, USA, July 2020. doi: 10.15607/RSS.2020.XVI.034.

[5] D. Seita, A. Ganapathi, R. Hoque, M. Hwang, E. Cen, A. K. Tanwani, A. Balakrishna, B. Thananjeyan, J. Ichnowski, N. Jamali, K. Yamane, S. Iba, J. Canny, and K. Goldberg. Deep Imitation Learning of Sequential Fabric Smoothing From an Algorithmic Supervisor. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[6] W. Yan, A. Vangipuram, P. Abbeel, and L. Pinto. Learning predictive representations for deformable objects using contrastive estimation. *Robotics: Science and Systems*, 2020.

[7] A. Nair, D. Chen, P. Agrawal, P. Isola, P. Abbeel, J. Malik, and S. Levine. Combining self-supervised learning and imitation for vision-based rope manipulation. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 2146–2153. IEEE, 2017.

[8] D. Seita, P. Florence, J. Tompson, E. Coumans, V. Sindhwani, K. Goldberg, and A. Zeng. Learning to rearrange deformable cables, fabrics, and bags with goal-conditioned transporter networks. *arXiv preprint arXiv:2012.03385*, 2020.

[9] R. Lee, D. Ward, A. Cosgun, V. Dasagi, P. Corke, and J. Leitner. Learning arbitrary-goal fabric folding with one hour of real robot experience. *Conference on Robot Learning*, 2020.

[10] Y. Wu, W. Yan, T. Kurutach, L. Pinto, and P. Abbeel. Learning to Manipulate Deformable Objects without Demonstrations. In *Proceedings of Robotics: Science and Systems*, Corvalis, Oregon, USA, July 2020. doi:10.15607/RSS.2020.XVI.065.

[11] A. Ganapathi, P. Sundaresan, B. Thananjeyan, A. Balakrishna, D. Seita, J. Grannen, M. Hwang, R. Hoque, J. E. Gonzalez, N. Jamali, et al. Learning dense visual correspondences in simulation to smooth and fold real fabrics. *arXiv preprint arXiv:2003.12698*, 2020.

[12] C. Smith, Y. Karayiannidis, L. Nalpantidis, X. Gratal, P. Qi, D. V. Dimarogonas, and D. Kragic. Dual arm manipulation—a survey. *Robotics and Autonomous Systems*, 60(10):1340–1353, 2012. ISSN 0921-8890. doi:https://doi.org/10.1016/j.robot.2012.07.005.

[13] A. Edsinger and C. C. Kemp. Two arms are better than one: A behavior based control system for assistive bimanual manipulation. In *Recent progress in robotics: Viable robotic service to human*, pages 345–355. Springer, 2007.

[14] R. L. A. Shauri and K. Nonami. Assembly manipulation of small objects by dual-arm manipulator. *Assembly Automation*, 2011.

[15] R. Suárez, J. Rosell, and N. García. Using synergies in dual-arm manipulation tasks. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5655–5661, 2015. doi:10.1109/ICRA.2015.7139991.

[16] R. Bordalba, L. Ros, and J. M. Porta. A randomized kinodynamic planner for closed-chain robotic systems. *IEEE Transactions on Robotics*, 2020.

[17] F. Xie, A. Chowdhury, M. C. De Paolis Kaluza, L. Zhao, L. Wong, and R. Yu. Deep imitation learning for bimanual robotic manipulation. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 2327–2337. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/18a010d2a9813e91907ce88cd9143fdf-Paper.pdf`.

[18] R. Chitnis, S. Tulsiani, S. Gupta, and A. Gupta. Efficient bimanual manipulation using learned task schemas. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1149–1155. IEEE, 2020.

[19] R. Chitnis, S. Tulsiani, S. Gupta, and A. Gupta. Intrinsic motivation for encouraging synergistic behavior. *International Conference on Learning Representations*, 2020.

[20] J. Sanchez, J. Corrales, B. Bouzgarrou, and Y. Mezouar. Robotic manipulation and sensing of deformable objects in domestic and industrial applications: a survey. *The International Journal of Robotics Research*, 37:688 – 716, 2018.

[21] I. Garcia-Camacho, M. Lippi, M. C. Welle, H. Yin, R. Antonova, A. Varava, J. Borras, C. Torras, A. Marino, G. Alenyà, and D. Kragic. Benchmarking bimanual cloth manipulation. *IEEE Robotics and Automation Letters*, 5(2):1111–1118, 2020. doi:10.1109/LRA.2020.2965891.

[22] J. Borràs, G. Alenyà, and C. Torras. A grasping-centered analysis for cloth manipulation. *IEEE Transactions on Robotics*, 36(3):924–936, 2020.

[23] Y. Li, D. Xu, Y. Yue, Y. Wang, S.-F. Chang, E. Grinspun, and P. K. Allen. Regrasping and unfolding of garments using predictive thin shell modeling. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[24] M. Cusumano-Towner, A. Singh, S. Miller, J. F. O'Brien, and P. Abbeel. Bringing clothing into desired configurations with limited perception. In *2011 IEEE International Conference on Robotics and Automation*, pages 3893–3900, 2011. doi:10.1109/ICRA.2011.5980327.

[25] D. Tanaka, S. Arnold, and K. Yamazaki. Emd net: An encode–manipulate–decode network for cloth manipulation. *IEEE Robotics and Automation Letters*, 3(3):1771–1778, 2018. doi:10.1109/LRA.2018.2800122.

[26] R. Jangir, G. Alenyà, and C. Torras. Dynamic cloth manipulation with deep reinforcement learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4630–4636. IEEE, 2020.

[27] H. Ha and S. Song. Flingbot: The unreasonable effectiveness of dynamic manipulation for cloth unfolding. *arXiv preprint arXiv:2105.03655*, 2021.

[28] S. Miller, J. V. D. Berg, M. Fritz, T. Darrell, K. Goldberg, and P. Abbeel. A geometric approach to robotic laundry folding. *The International Journal of Robotics Research*, 31:249 – 267, 2012.

[29] Y. Li, Y. Wang, M. Case, S.-F. Chang, and P. K. Allen. Real-time pose estimation of deformable objects using a volumetric approach. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1046–1052, 2014. doi:10.1109/IROS.2014.6942687.

[30] C. Chi and S. Song. Garmentnets: Category-level pose estimation for garments via canonical space shape completion. *arXiv preprint arXiv:2104.05177*, 2021.

[31] D. Pathak, P. Mahmoudieh, G. Luo, P. Agrawal, D. Chen, Y. Shentu, E. Shelhamer, J. Malik, A. A. Efros, and T. Darrell. Zero-shot visual imitation. In *ICLR*, 2018.

[32] M. Lippi, P. Poklukar, M. C. Welle, A. Varava, H. Yin, A. Marino, and D. Kragic. Latent space roadmap for visual action planning of deformable and rigid object manipulation. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5619–5626. IEEE, 2020.

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

[34] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2758–2766, 2015.

[35] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2462–2470, 2017.

[36] Z. Teed and J. Deng. Raft: Recurrent all-pairs field transforms for optical flow. In *European Conference on Computer Vision*, pages 402–419. Springer, 2020.

[37] S. Dong, D. K. Jha, D. Romeres, S. Kim, D. Nikovski, and A. Rodriguez. Tactile-rl for insertion: Generalization to objects of unknown geometry. *arXiv preprint arXiv:2104.01167*, 2021.

[38] A. Amiranashvili, A. Dosovitskiy, V. Koltun, and T. Brox. Motion perception in reinforcement learning with dynamic objects. In *Conference on Robot Learning*, pages 156–168. PMLR, 2018.

[39] K. Yamazaki, R. Oya, K. Nagahama, K. Okada, and M. Inaba. Bottom dressing by a dual-arm robot using a clothing state estimation based on dynamic shape changes. *International Journal of Advanced Robotic Systems*, 13(1):5, 2016. doi:10.5772/61930.

[40] M. Argus, L. Hermann, J. Long, and T. Brox. Flowcontrol: Optical flow based visual servoing. *arXiv preprint arXiv:2007.00291*, 2020.

[41] X. Lin, Y. Wang, J. Olkin, and D. Held. Softgym: Benchmarking deep reinforcement learning for deformable object manipulation. In *Proceedings of (CoRL) Conference on Robot Learning*, November 2020.

[42] S. Chitta, I. Sucan, and S. Cousins. Moveit![ros topics]. *IEEE Robotics & Automation Magazine*, 19(1):18–19, 2012.

[43] C. G. Harris and M. Stephens. A combined corner and edge detector. In *Alvey Vision Conference*, 1988.

[44] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Supplement Contents

A Additional Details and Results for FabricFlowNet

A.1 FabricFlowNet Implementation Details

Data Collection. We collect data in SoftGym by taking random pick and place actions on the cloth. The random actions are biased to pick corners of the cloth mask (detected using Harris corner detection [43]) 45% of the time, and "true" corners of the square cloth 45% of the time. If the true corners are occluded then Harris corners are used instead. For the remaining 10%, the pick actions are uniformly sampled over the visible cloth mask. After the pickers grasp the cloth, they lift to a fixed height of 7.5 cm.

We constrain the place points of the action so that both place points are offset in the same direction and distance from their respective pick points. The direction is orthogonal to the segment connecting the two pick points, and points towards the center of the image, so the cloth does not move out of the frame (similar to Lee *et al.* [9]). The distance between the pick point and the place point along this direction is uniformly sampled between $[25, 150]$ px. The distance is truncated if it exceeds a margin of 20 px from the image edge, again to prevent moving the cloth out of the frame. While these heuristics may seem to overly constrain the data we collect, we observe that our data still contains highly diverse cloth configurations, as shown in Fig. S1.

For each sample, we save the initial depth observation image, the dual-arm pick and place pixel locations of the action, the next depth observation resulting from the executed action, and the cloth

particle positions of both observations (See Fig. S1). The camera for capturing depth observations is fixed at 65 cm above the support surface. We mask the depth observations to only include the cloth by setting all background pixels to zero. The dataset for training both the flow and pick networks consists of 20k samples from 4k episodes, where each episode consists of five dual-arm pick and place actions.

Flow Network Training. We use FlowNet [35] as our flow network architecture. The input to FlowNet is the initial and next depth image from a sample in our dataset, stacked channel-wise. The ground truth flow for supervising FlowNet comes from the cloth particles used by the simulator to model the cloth's dynamics: we collect the cloth particle positions for each observation in our dataset and correspond them across observations to get flow vectors (See Fig. S1). The ground truth flow is sparse because the cloth particles are sparse, so we train FlowNet using a masked loss that only includes pixels with corresponding ground truth flow. Similar to Lee *et al.* [9], we apply spatial augmentation of uniform random translation (up to 5 px) and rotation (up to 5 degrees) to augment the training data. We train the network using the dataset of 20k random actions described above. We use the Adam [44] optimizer, learning rate 1e-4, weight decay 1e-4, and batch size 8.

PickNet Network Training. PickNet1 and PickNet2 are fully-convolutional network architectures based on Lee *et al.* [9], with 4 convolutional layers in the encoder, each with 32 filters of size 5. The first three layers of the encoder have stride 2 and the last one has stride 1. The decoder consists of 2 interleaved convolutional layers and bilinear upsampling layers.

The input to the PickNet1 is a 200 $\times$ 200 flow image. PickNet2 receives the first pick point location (the argmax of the Picknet1 output, as described in the main text) as an additional input, represented as a 2D Gaussian $\mathcal{N}(p_1, \sigma)$ (where $\sigma = 5$). Similar to Nair *et al.* [7], the output of both networks is a 20 x 20 spatial grid. If the pick points predicted by PickNet are not on the cloth mask, we project them to the closest pixel on the mask using an inverse distance transform. In practice, we find that the predictions are usually either on the cloth mask or very close to the mask. To train PickNet1 and PickNet2, we use the same dataset of 20k random actions described above. We use the Adam [44] optimizer, learning rate 1e-4, and batch size 10.

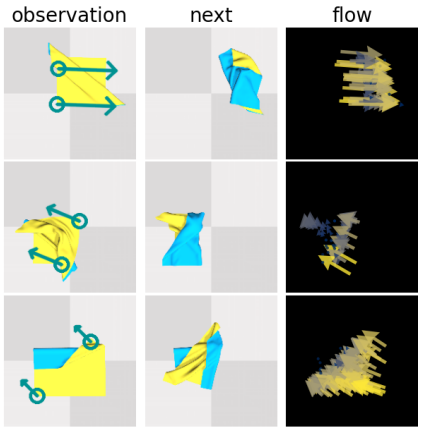


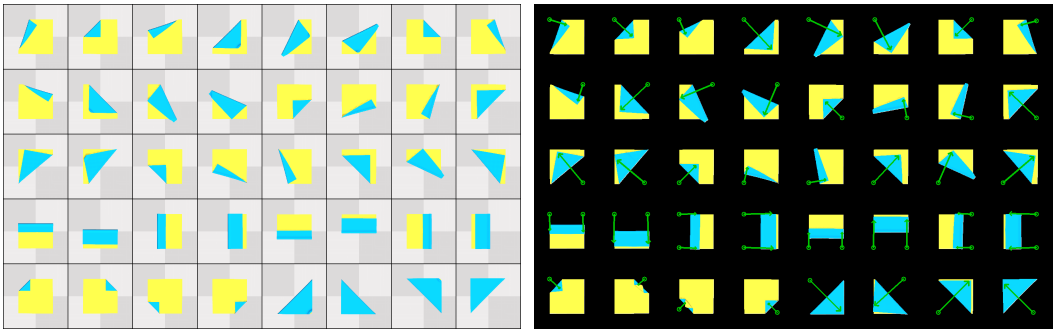
Figure S1: Training data for FFN.

A.2 Additional Simulation Results for FabricFlowNet

Fig. S2b and Fig. S2f show the cloth configurations achieved by FabricFlowNet for each of the one-step goals (Fig. S2a) and multi-step goals (Fig. S2e). Fig. S1 provides examples of the data used to train FFN. Our policy is deterministic and the simulation is near-deterministic, so we only need 1 trial for our simulation experiments (unlike our real world experiments which use 3 trials).

A.3 Additional Real World Details and Results for FabricFlowNet

Cloth Masking. In simulation, we can obtain a perfect cloth mask. In the real world, we first obtain a background mask of the table using color-based HSV thresholding, which we can determine before the cloth is placed on the table. We then use the inverse of this background mask to obtain a mask of



(a) One-step goals

(b) One-step FFN performance

(c) One-step Fabric-VSF performance

(d) One-step Lee et al. performance

(e) Multi-step goals

(f) Multi-step FFN performance

(g) Multi-step Fabric-VSF performance

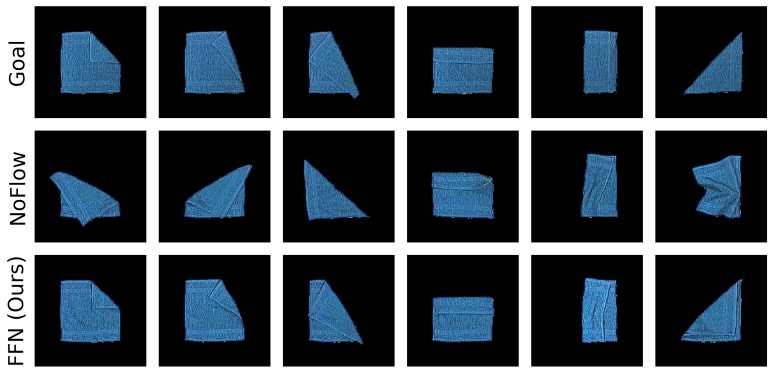
(h) Multi-step Lee et al. performance

Figure S2: Goal configurations, achieved configurations, and training data in simulation. Arrows indicate the executed action. Fabric-VSF uses a lower camera height than FFN (45 cm vs. 65 cm), thus the cloth looks slightly larger.

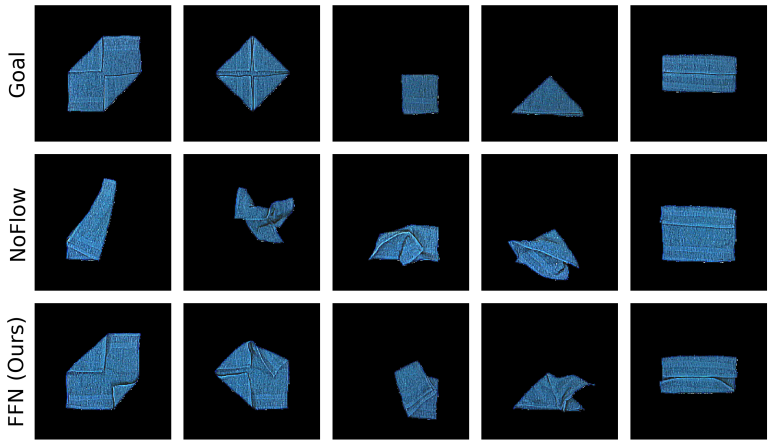
the cloth. Note that while we use background color of the table for cloth masking, the network itself only takes depth input, allowing the network to be robust to colors and patterns on the cloth itself.

Results on Real Cloth Folding. Table S1 provides mean IOU (mIOU) performance for NoFlow and FFN on real cloth goals. The NoFlow ablation performs considerably worse compared to FFN on real cloth folding. Qualitative results and the complete set of real square cloth goals are in Fig. S3; the complete set of real rectangle and T-shirt goals are in the main text. We found that for FFN, using FlowNet weights from epochs at the start of convergence transferred better to the real world than using weights from epochs long after convergence.

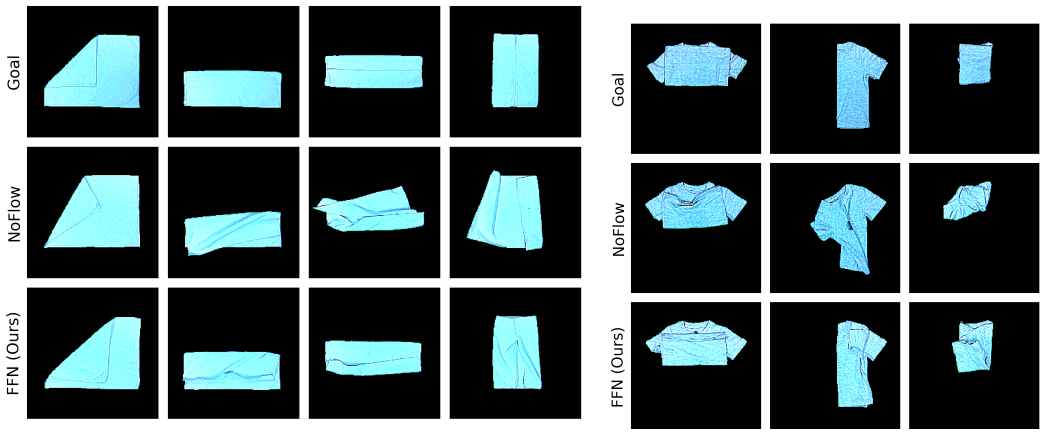
Failure Cases. This work focused on high level actions with fixed primitives for picking and placing that may not be ideal for all cloth types, sizes, or folds. Causes of failures include the grasped portion of the cloth "flopping back" against the folding direction, undoing small folding actions



Single-step Square Towel



Multi-step Square Towel



Rectangular Cloth T-shirt

Figure S3: Qualitative performance of FFN and NoFlow on real cloth. The trial corresponding to the best achieved IOU is shown for each example. For multi-step goals, only the final goal is shown. FFN only takes depth images as input, allowing it to easily transfer to cloths of different colors. Contrast and brightness have been adjusted to enhance visibility.

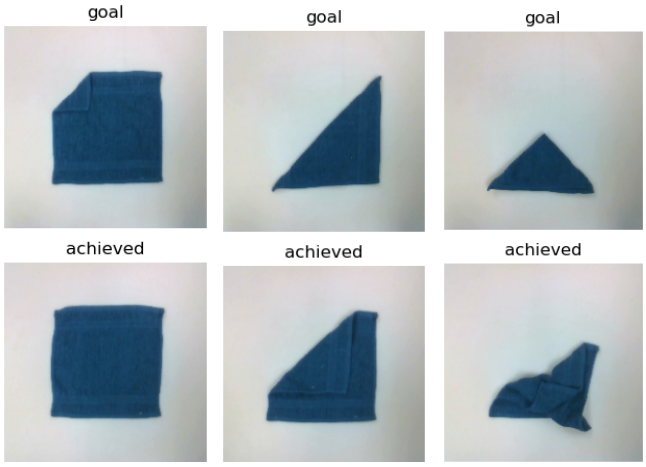
or causing unwanted secondary folds (Fig. S4a). Potential future work is to learn better pick and place primitives. Another source of failure was over- or under-estimating the fold distance due to slight inaccuracies in the flow prediction (Fig. S4b). We also see some failures during multi-

Table S1: mIOU for Folding Square Towel, Rectangular Cloth, and T-shirt

Method	1-Step Sq. ↑ ($n = 6$)	Multi-Step Sq. ↑ ($n = 5$)	All Sq. ↑ ($n = 11$)	Rect. ↑ ($n = 3$)	T-shirt ↑ ($n = 3$)
NoFlow	0.59 ± 0.04	0.45 ± 0.01	0.53 ± 0.02	0.65 ± 0.07	0.61 ± 0.06
FFN (Ours)	$\mathbf{0.89 \pm 0.01}$	$\mathbf{0.69 \pm 0.04}$	$\mathbf{0.80 \pm 0.03}$	$\mathbf{0.81 \pm 0.04}$	$\mathbf{0.82 \pm 0.02}$

Average of 3 rollouts. Higher mIOU scores are better; the max achievable score is 1.0.

step folding; since we provide sub-goals in sequence and allow only one action per sub-goal, the discrepancy between the starting image of the demonstration and the observed image can result in poor predictions (Fig. S4c). Allowing the policy to take multiple actions to achieve a sub-goal before proceeding may improve performance. For example, the flow can be recalculated after each action to determine if the observation is sufficiently close to the desired sub-goal configuration before proceeding to the next sub-goal.



(a) Flopping back (b) Undershooting (c) Poor prediction

Figure S4: Examples of failure cases

B Additional Details and Results for Fabric-VSF [4]

B.1 Fabric-VSF [4] Implementation Details

The original Fabric-VSF [4] paper uses single arm actions and a top-down close camera view such that the cloth covers the whole image. To match the camera view, we set the camera height to be 45 cm above the table in our case. The training dataset consists of 7115 trajectories, each with 15 random pick-and-place actions, totaling 106725 data points. Note that this dataset is 5x larger than the 20k samples we train FFN on. During training, Fabric-VSF takes as input 3 context frames and predicts the next 7 target frames.

We trained 8 variants of Fabric-VSF. Each variant differs in the following aspects: 1) whether it uses single arm or dual arms; 2) during data collection, whether the pick-and-place actions are randomly sampled, or use the corner biasing sampling strategy as described in Sec. A.1, and 3) whether it uses the original small action size ("Small Action", bounded to half of the cloth width) or a larger action size ("Large Action", bounded to the diagonal length of the cloth). Other than these three changes, we set all other parameters to be the same as in the original paper. Therefore, the variant with single arm actions, no corner biasing during data collection, and small action size is exactly how Fabric-VSF is trained in the original paper.

After the training, we plan with cross-entropy method (CEM) to find actions for achieving a given goal image. We use the exact same CEM parameters as in the original paper, *i.e.,* we run CEM for 10 iterations, each with a population size of 2000 and elite size of 400.

B.2 Additional Fabric-VSF [4] Results

The results for the Fabric-VSF variants are summarized in Table S2. We note that the variant using single arm actions, corner biasing for data collection, and large action size performs the best out of all variants. This variant outperforms FFN on overall error and one-step error, but performs slightly worse than FFN on multi-step error (See Fig. S2c and Fig. S2g for qualitative results). However, we note that Fabric-VSF was trained on 5x more data than FFN. Additionally, Fabric-VSF takes much longer to run at inference time, requiring $\sim$7 minutes of CEM iterations to compute a single action compared to $\sim$0.007 seconds for a forward pass through FFN. 7 minutes of CEM planning time is impractical for real-world folding. We also demonstrate in the following section that FFN generalizes to other cloth shapes better than Fabric-VSF.

Analyzing the performance between different Fabric-VSF variants, for single-arm actions, using large actions instead of small actions always leads to better performance. However, this is not true for the dual arm variants. Interestingly, we find that using dual arms tends to result in worse performance compared with using a single arm. The reason for this could be that during CEM planning, dual-arm variants double the action dimension, which increases complexity for CEM and makes it difficult to find optimal actions.

Table S2: Mean Particle Distance Error (mm) and Inference Time (sec) for Fabric-VSF Variants

Baseline	1-Step (n=40)	Multi-Step (n=6)	All (n=46)	Inf. Time
1-Arm, No CB, Sm. Action	12.92 ± 13.00	46.05 ± 48.07	17.24 ± 23.93	$\sim$420s
1-Arm, No CB, Lg. Action	10.13 ± 07.33	33.06 ± 12.46	13.12 ± 11.25	$\sim$420s
1-Arm, CB, Sm. Action	14.09 ± 11.36	38.68 ± 27.72	17.30 ± 16.76	$\sim$420s
1-Arm, CB, Lg. Action	6.30 ± 06.55	$\mathbf{21.33 \pm 11.20}$	8.27 ± 08.90	$\sim$420s
2-Arm, No CB, Sm. Action	24.60 ± 14.69	50.26 ± 27.54	27.94 ± 19.00	$\sim$420s
2-Arm, No CB, Lg. Action	10.98 ± 05.80	40.92 ± 18.06	14.89 ± 13.17	$\sim$420s
2-Arm, CB, Sm. Action	16.21 ± 13.81	36.42 ± 26.51	18.84 ± 17.43	$\sim$420s
2-Arm, CB, Lg. Action	15.58 ± 10.88	54.06 ± 26.68	20.60 ± 19.07	$\sim$420s
FFN (Ours)	$\mathbf{4.46 \pm 02.62}$	25.04 ± 22.88	$\mathbf{7.14 \pm 11.06}$	$\sim$**0.007s**

CB: Corner Bias Sm. Action: Small Action Lg. Action: Large Action

C Additional Details and Results for Lee *et al.* [9]

C.1 Lee *et al.* [9] Implementation Details

Lee *et al.* [9] learns a fabric folding policy for a discrete action space using a fully convolutional state-action value function, or Q-network. Observation and goal images are stacked channel-wise, then duplicated and transformed to form a batch of m image rotations and n scales to represent different pick and place directions and action lengths. The whole batch is input to the Q-network to compute the Q-value of executing an action for each rotation and scale at every point on the image. The action corresponding to the max Q-value from the outputs is executed. The discrete action space of m rotations and n action lengths for Lee *et al.* [9] enables efficient policy learning, but greatly limits the actions of the learned policy compared to FFN.

We extend Lee *et al.* [9] from a single-arm approach to a dual-arm one. To represent two pickers instead of one, we input two pairs of observation and goal images to the Q-network. When rotating and scaling the images to represent different actions, the images are constrained to have the same rotation, but are allowed to be scaled differently. In other words, the dual-arm actions are constrained to execute pick and place actions in the same direction, but can have different pick and place lengths. The Q-network outputs a pair (one for each arm) of Q-value heatmaps for every action in the discrete action space (*i.e.,* every rotation and scale). The max Q-value in each of the two heatmaps is aver-

aged, and the heatmap pair with the highest averaged Q-value is selected from the set of all discrete rotations and scales. The picker action corresponding to the argmax of each heatmap is executed.

We train each Lee *et al.* variant below using hyperparameters similar to the original paper [9], training for 25k steps with learning rate 1e-4, batch size 10, and evaluating performance on test goals every 500 steps to find the best performing step.

C.2 Additional Lee *et al.* [9] Results

We trained variants of Lee *et al.* to compare single-arm vs. dual-arm performance, depth input vs. RGB input, collecting data with corner bias similar to FFN vs. without bias, and using the original close-up image of the cloth ("Low Cam") vs. images from further away ("High Cam"). All variants were trained with 20k training examples. We also provide results for two variants of FFN trained on the same amount of data, one where actions are sampled from the discrete action space (*i.e.,* discretized action angles and lengths) in Lee *et al.* [9] ("Discrete Actions"), and the other where actions are sampled using our continuous action space described in Sec. A.1 ("Cont. Actions"). Lee *et al.* [9] is an inherently discrete approach and cannot be trained to output continuous actions, nor can it be trained on data with actions outside of its discrete action space.

Table S3 shows that the performance of all Lee *et al.* variants is poor compared to FFN, particularly on 1-step goals (see Appendix Fig. S2d and Appendix Fig. S2h for qualitative results). FFN outperforms Lee *et al.* when trained on either the discrete action dataset or the continuous one. Training FFN on continuous actions results in better performance for 1-step goals, but the discrete action dataset also performs fairly well. These results indicate that the improved performance of FFN vs. Lee *et al.* cannot be solely explained by training on continuous vs. discrete action data, though other factors like outputting continuous actions instead of discrete ones may still play significant role in FFN's improved performance.

Table S3: Mean Particle Distance Error for Lee *et al.* on 20k Training Examples

Baseline	1-Step (40)	Multi Step (6)	All (46)
Lee *et al.*, 1-Arm, D, No CB, LC	18.94 ± 16.43	24.18 ± 17.75	19.62 ± 16.49
Lee *et al.*, 1-Arm, D, No CB, HC	16.18 ± 08.38	26.20 ± 16.31	17.49 ± 10.10
Lee *et al.*, 1-Arm, D, CB, LC	20.99 ± 18.88	34.61 ± 31.35	22.77 ± 20.97
Lee *et al.*, 1-Arm, D, CB, HC	19.70 ± 09.37	38.91 ± 24.05	22.20 ± 13.53
Lee *et al.*, 1-Arm, RGB, No CB, LC	49.29 ± 18.10	52.03 ± 33.62	49.65 ± 20.26
Lee *et al.*, 1-Arm, RGB, No CB, HC	47.12 ± 21.04	64.48 ± 29.85	49.38 ± 22.75
Lee *et al.*, 1-Arm, RGB, CB, LC	33.89 ± 19.01	58.90 ± 43.34	37.15 ± 24.38
Lee *et al.*, 1-Arm, RGB, CB, HC	39.01 ± 25.36	55.46 ± 38.38	41.15 ± 27.43
Lee *et al.*, 2-Arm, D, No CB, LC	36.62 ± 14.51	47.72 ± 21.95	38.07 ± 15.82
Lee *et al.*, 2-Arm, D, No CB, HC	40.75 ± 13.22	52.88 ± 19.03	42.33 ± 14.45
Lee *et al.*, 2-Arm, D, CB, LC	47.18 ± 18.60	57.29 ± 28.65	48.50 ± 20.07
Lee *et al.*, 2-Arm, D, CB, HC	35.98 ± 24.60	64.75 ± 51.76	39.73 ± 30.30
FFN, 2-Arm, D, CB, HC, Discrete Actions	9.57 ± 06.07	$\mathbf{10.15 \pm 07.20}$	10.17 ± 07.34
FFN, 2-Arm, D, CB, HC, Cont. (Ours)	$\mathbf{4.46 \pm 02.62}$	25.04 ± 22.88	$\mathbf{7.14 \pm 11.06}$

D: Depth CB: Corner Bias LC: Low Camera HC: High Camera Cont: Continuous Actions

Lee *et al.* with and without Subgoals. FFN uses subgoals at inference time in order to fully specify the task; many cloth folding goals have final goal configurations in which large portions of the cloth are self-occluded. Subgoals are required to ensure the task is completed correctly and that the cloth is correctly folded. Lee *et al.* [9] demonstrated cloth folding without subgoals at inference time by relying on a learned Q-value heatmap to select actions toward a final end goal. We compare the performance of the best Lee *et al.* variant with and without subgoals at test-time. The results of this experiment are in Table S4. While the performance on 1-step goals are similar because those tasks do not have subgoals, performance on multi-step goals is worse without subgoals.

Table S4: Mean Particle Distance Error for Lee *et al.* With and Without Subgoals

Method	1-Step (40)	Multi Step (6)	All (46)
Lee *et al.*	16.92 ± 9.28	37.74 ± 38.99	19.71 ± 20.27
Lee *et al.*, With Subgoals	$\mathbf{16.18 \pm 8.38}$	$\mathbf{26.20 \pm 16.31}$	$\mathbf{17.49 \pm 10.10}$

D Additional Details and Results for Ablations

D.1 Ablation Implementation Details

NoFlowIn The architecture for this ablation is identical to our main method, except that it takes depth images instead of flow images as input. We use a conditioned architecture with two PickNets; PickNet1 receives the observation and goal depth images as input both of size 200×200. The place point is computed by querying the flow image similar to our main method.

NoFlowPlace We predict the place points similarly to the pick points by using an additional place network. The place network architecture is identical to PickNet. The input is a flow image and the output is the place point predictions.

NoFlow This ablation is a combination of NoFlowIn and NoFlowPlace, where PickNet and PlaceNet both take observation and goal depth images as input.

NoCornerBias This ablation is the same as our main method except for the training dataset. We use a dataset that does not bias the data to pick corners (See Sec. A.1). Instead, the pick actions are always uniformly sampled over the visible cloth mask. We still constrain the folding actions for both arms to be in the same direction and distance from their respective pick points and point towards the center of the frame.

NoSplitPickNet The architecture of PickNet is modified so that we only have one PickNet for both arms instead of the conditioned architecture used in our main method. The PickNet takes as input the flow image and outputs two heatmaps corresponding to the two pick points.

NoMinLoss The loss in Eq. 1 is replaced with the following:

$$\mathcal{L}_{\text{NoMin}} = \text{BCE}(H_1, H_1^*) + \text{BCE}(H_2, H_2^*) \tag{2}$$

D.2 Additional Ablation Results

We provide ablation results in Table S5 grouped by single-step, multi-step, and all goals.

Table S5: Mean Particle Distance Error for Ablations

Ablation	One Step (n=40)	Multi Step (n=6)	All (n=46)
NoFlowIn	5.14 ± 3.62	24.63 ± 21.30	9.37 ± 12.20
NoFlowPlace	7.61 ± 5.44	30.25 ± 17.62	10.56 ± 11.15
NoFlow	8.97 ± 7.45	28.79 ± 19.33	18.02 ± 20.34
NoCornerBias	9.79 ± 5.57	$\mathbf{19.61 \pm 17.52}$	11.07 ± 8.83
NoSplitPickNet	4.87 ± 2.61	23.41 ± 18.87	7.29 ± 9.56
NoMinLoss	5.10 ± 4.04	20.81 ± 17.57	7.15 ± 9.08
FFN (Ours)	$\mathbf{4.46 \pm 02.62}$	25.04 ± 22.88	$\mathbf{7.14 \pm 11.06}$

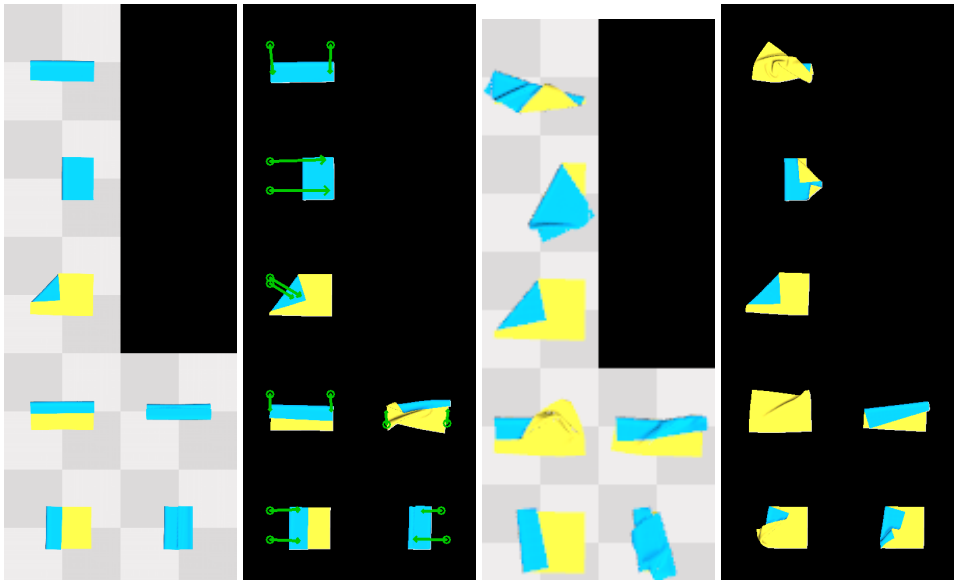
E Additional Results on Unseen Cloth Shapes

We also evaluate Fabric-VSF and Lee *et al.* on generalization to unseen cloth shapes. FFN generalizes well to new shapes, as shown in the main text (see Fig. 5 and Sec. 4.2.1). Table S6 provides quantitative results on the rectangle cloth and T-shirt for the best Fabric-VSF method and best

Lee *et al.* method compared to FFN. FFN outperforms both methods by a large margin. Fabric-VSF generalizes poorly, likely because it relies on planning with a learned visual dynamics model. Lee *et al.* also does not generalize well compared to FFN. Fig. S5 provides a qualitative comparison.

Table S6: Mean Particle Distance for Folding Unseen Cloth Shapes in Simulation

Method	Rectangle (n=6)	T-Shirt (n=3)
Lee *et al.* , 1-Arm, No Corner Bias, High Cam, 20k Actions	31.63 ± 18.04	86.65 ± 34.67
Fabric-VSF, 1-Arm, Corner Bias, Large Action	25.68 ± 11.21	45.25 ± 13.83
FFN (Ours)	$\mathbf{10.70 \pm 08.54}$	$\mathbf{20.91 \pm 11.28}$



(a) Rect. cloth goals (b) FFN achieved (c) Fabric-VSF achieved (d) Lee *et al.* achieved

Figure S5: Qualitative performance of FFN, Fabric-VSF, and Lee *et al.* on rectangular cloth.

F End-to-End Variants of FFN

We investigate the effect of training our FFN architecture end-to-end. First, we train the FFN architecture with pick losses as well as the flow loss; all losses are allowed to backpropagate through the entire combined network, including through the FlowNet layers. The results on the square towel are presented in Table S7 ("JointFFN"). This variant performs significantly worse than FFN (9.28 vs. 7.14 on all goals).

Table S7: Mean Particle Distance Error (mm) for End-to-End Variants of FFN

Method	1-Step (n=40)	Multi-Step (n=6)	All (n=46)
JointFFN	07.60 ± 05.62	$\mathbf{17.53 \pm 15.56}$	09.28 ± 09.39
JointPredictPlace	12.90 ± 11.67	35.25 ± 19.22	22.88 ± 23.24
JointFFN, No Flow Loss	32.41 ± 22.61	68.17 ± 50.35	37.07 ± 30.34
JointPredictPlace, No Flow Loss	16.31 ± 22.73	50.27 ± 31.44	24.39 ± 29.77
FFN (Ours)	$\mathbf{4.46 \pm 02.62}$	25.04 ± 22.88	$\mathbf{7.14 \pm 11.06}$

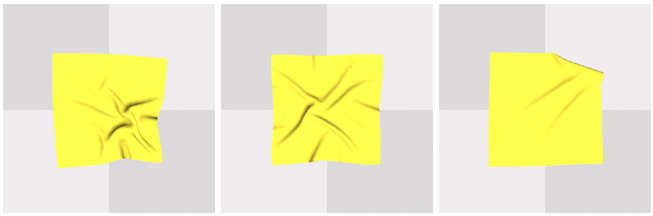
We also trained another variant which consists of a FlowNet, a PickNet, and a PlaceNet, trained end-to-end ("JointPredictPlace" in Table S7). This is similar to our ablation "PredictPlace" in Table 2, which uses the same architecture but is not trained end-to-end. JointPredictPlace performs significantly worse than FFN (22.88 vs. 7.14 on all goals) and also underperforms compared to Predict-

Place (10.56 on all goals). Overall, this result, as well as the one in the paragraph above, indicate that end-to-end training leads to significantly worse performance for this task. Our intuition for this is that the flow network should be trained only with the flow loss, and that backpropagating the gradients from the pick loss into the flow network adds noise and reduces its performance.

Lastly, we evaluated variants of the above two architectures with the flow loss removed, to see if we could train these architectures end-to-end with just a single loss at the end, instead of using an intermediate flow loss. The results, shown in Table S7, are worse for both variants, showing the importance of the intermediate flow loss.

G FFN Performance with Crumpled Starting Configurations

Our experiments focused on folding tasks, and we assume that a previous method was used to flatten the cloth before our method is executed. To evaluate the robustness of our method to imperfect smoothing, we evaluate the performance of FFN in simulation on slightly crumpled initial cloth configurations. We generated crumpled configurations by taking the flat cloth and executing a random pick and place action with a maximum translation of 10 pixels. The three configurations used in our experiments are shown in Fig. S6.



(a) Crumpled 0 (b) Crumpled 1 (c) Crumpled 2

Figure S6: Crumpled initial cloth configurations

For each crumpled configuration, we evaluated FFN on the full set of 46 evaluation goals, where the starting configuration of the cloth was set to the given crumpled configuration. The results of these evaluations are in Table S8. The particle distance error is slightly higher with the crumpled starting configurations, but the qualitative results in Fig. S7 show that FFN still produces actions that are very close to the intended goals.

Table S8: Mean Particle Distance Error (mm) for FFN with Different Start Configurations

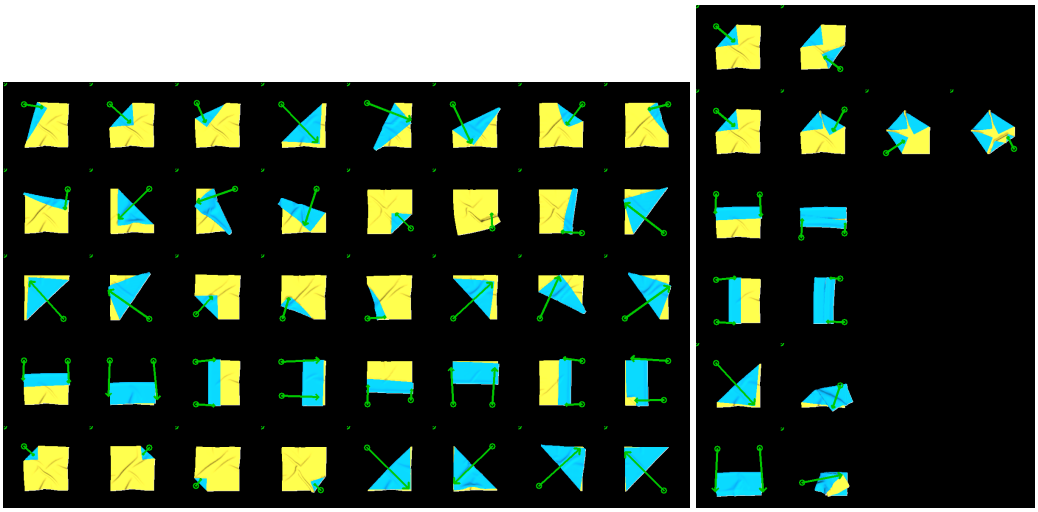
Starting Config	1-Step (n=40)	Multi-Step (n=6)	All (n=46)
FFN, Crumpled 0	12.40 ± 4.82	24.82 ± 24.81	14.01 ± 10.86
FFN, Crumpled 1	10.68 ± 2.89	23.54 ± 22.56	12.36 ± 9.61
FFN, Crumpled 2	10.68 ± 4.29	$\mathbf{21.05 \pm 14.70}$	12.03 ± 7.51
FFN, Flat	$\mathbf{4.46 \pm 2.62}$	25.04 ± 22.88	$\mathbf{7.14 \pm 11.06}$

H FFN Performance with Iterative Refinement

Table S9: Mean Particle Distance Error (mm) for FFN with Iterative Refinement

Starting Config	1-Step (n=40)	Multi-Step (n=6)	All (n=46)
FFN, No Refinement	$\mathbf{4.46 \pm 2.62}$	25.04 ± 22.88	7.14 ± 11.06
FFN, Iterative Refinement	4.54 ± 2.58	$\mathbf{20.47 \pm 19.49}$	$\mathbf{6.62 \pm 9.17}$

In our normal evaluations, each goal or subgoal is attempted only once by each method. With a single attempted action for each subgoal, FFN is able to achieve a diverse set of goals, as demonstrated in this work. However, we find that FFN can achieve even better performance when attempting goals



(a) Crumpled one-step FFN performance

(b) Crumpled Multi-step FFN performance

Figure S7: Configurations achieved by FFN when starting from the "Crumpled 1" configuration for each attempt (compare with Fig. S2)

multiple times, using the flow to compare the current observation with the goal and taking actions that move the observation closer to the goal if it has not yet been reached. We evaluate the benefit of using this "iterative refinement" procedure in simulation. FFN moves to the next subgoal when a minimum threshold for the average flow is achieved, so the flow acts as a goal recognizer. The policy is allowed a maximum of 3 iterative actions per subgoal to limit potential divergence. The results in Table S9 show that iterative refinement can improve performance, particularly on multi-step goals, where reaching the current subgoal accurately is important for achieving subsequent goals.

I FlowNet Performance

FlowNet achieves an average endpoint error (EPE) of 1.0268 on the set of simulated test goals. The test goals are not seen during training. Fig. S8 provides qualitative examples of FlowNet performance on simulated test goals.

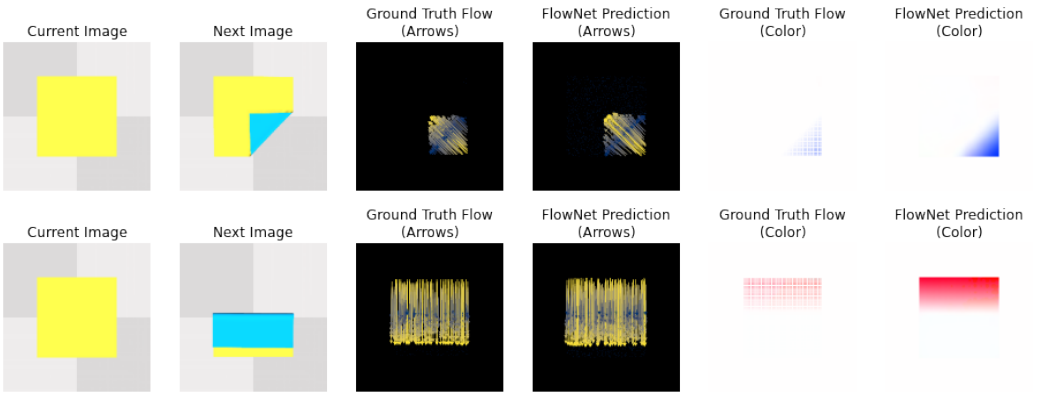


Figure S8: FlowNet Qualitative Performance. Two types of visualizations are provided: representing the flow vector as arrows, and representing the flow vector using RGB channels. FlowNet outputs a dense flow image but is trained on sparse ground truth flow. FlowNet takes only depth images as input; RGB images are shown as a visual aid only.