Robotic Laundry Folding

Evan Honnold
Dartmouth College

Robotic Laundry Folding

Senior Honors Thesis by Evan Honnold

Dartmouth Computer Science Technical Report TR2018-852

1 Introduction

The problem of robotic laundry folding is interesting because it offers the automation of a tedious everyday task, and also because it can provide insight into various other problems in robotics and computer vision. Current robotic systems are good at performing a precisely-defined sequence of actions many times, but have trouble receiving high-level instructions or adapting behavior to new situations. Our approach aims to address these two weaknesses: we automate low-level motion planning decisions, allowing our robot to rely on a few simple instructions; and we plan our folds using a stacked polygon model that generalizes easily to clothes of many shapes and sizes.

Our two main contributions are a “bar-folding” method for folding shirts that have already been unwrinkled and positioned on a table, and a planner for generating new sequences of folds that this method can use. The “bar-folding” method is quick, accurate, and simple compared to other folding methods, and the fold planner provides an automated alternative to the pre-programmed fold sequences used in other research.

2 Related Work

Hamajima and Kakikura divide the process of processing laundry into a “pipeline” composed of distinct, consecutive steps [5]. From a pile of miscellaneous crumpled clothes, the robot grabs a single item and classifies it as a shirt, pants, sock, or other garment. Next, the robot flattens the garment onto a table or other surface. Once the unwrinkled garment is in a predictable place, the robot folds it into a neat square or rectangular shape. Our work focuses on the final step of this pipeline – folding – but borrows ideas from the other steps, as well as from theoretical work on the immobilization of flexible objects.

Classification, Grabbing, and Flattening

Computer vision allows items to be identified and grabbed from a pile of laundry. Willimon *et al.* photograph the pile, segment the image by color, and grab whichever region is highest [6]. Once the garment is hanging from a gripper, Kita *et al.* fit it with a mass-spring model and simulate the effects of re-grabbing in various other places [7]. Doumanoglou *et. al.* improve the re-grabbing process with their “active random forests” recognition technique, and they manage to fully unfold the garment in the air before placing it on a table [3]. Other researchers have made various modifications (see Doumanoglou *et al.*'s survey), but the general approach is to use complex probabilistic models and repeated trials to re-grasp the garment into a recognizable shape.

Folding Flattened Clothes

Once the garment is flattened on the table, complex models are no longer necessary. van den Berg *et al.* model the garment as a two-dimensional polygon (as it would appear from a bird's-eye view above the table), perform a fold that creases the garment along a pre-determined “fold line,” and use their polygonal model to predict the new two-dimensional shape of the garment [1]. Doumanoglou *et al.* do the same during the folding step of their "pipeline," except they use computer vision to improve the polygonal model of the garment after each fold [3]. Both teams perform the folds using a strategy that imitates human motion: their two-armed robot grasps two edges or corners of the garment on one side of the fold line, lifts those edges and corners across the fold line while the rest of the garment remains on the table, and then sets the raised part down so the fold occurs at the line. Figures 1 and 2 demonstrate this strategy.

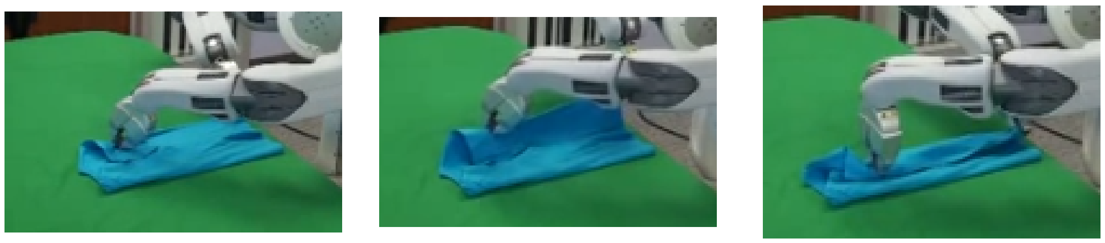


Figure 1: van den Berg *et al.*'s robot folding a shirt [9]

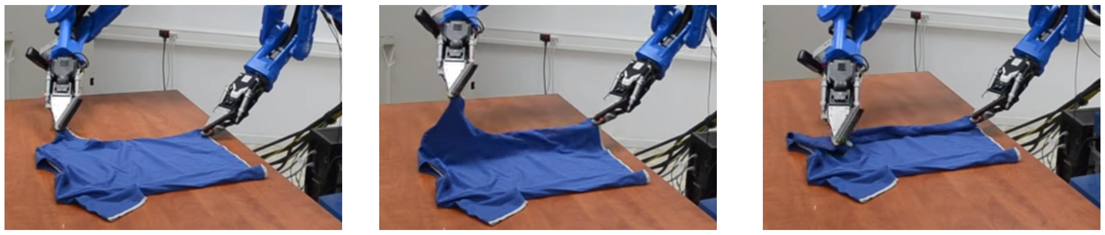


Figure 2: Doumanoglou *et al.*'s robot folding a shirt [10]

Immobilizing Flexible Objects

The folding strategy above requires the robot to immobilize the shirt during the lift motion. In the second image of Figure 2, for example, if the left gripper had not grabbed the end of the sleeve, then the sleeve would have sagged and been folded incorrectly. van den Berg *et al.* explain that their immobilization method rests on the theoretical foundation provided by Bell and Balkcom in their study of the number of "fingers" (i.e., grasp points) required to pin down a flexible polygon [4]. Bell's Ph.D. thesis discusses flexible-object immobilization in more detail, and also introduces the "Japanese method" for automated clothes-folding that served as the starting point of our work [2].

3 Japanese Clothes-Folding Method

The "Japanese Method" is a simple, quick way to fold T-shirts; see pages 30 through 34 of [2] for a detailed description. In his Ph.D. thesis, Matthew Bell automated this method by observing that the robot must grasp only three points on the shirt, none of which must be released until after the folding is complete. We began our work by replicating Bell's method on our six-degree-of-freedom arm. Figure 6 shows several stages of the process, and the video is available here. The shirt attaches to the bar because it contains several strategically placed magnets; we would use small grippers instead, if we were to develop the method further.

Our automated Japanese clothes-folding method is impractical: it generalizes poorly to other shapes and sizes of garment, and the arm configurations that define its movements were hard-coded. However, it provided us with an important idea: that with the help of a bar-shaped fixture, a single arm can perform folds that might seem to require two arms and grippers. We used this idea to create a "bar-folding" method that can perform many of the same folds that a human can, but without imitating the two-armed, grabbing-based strategies that we use for our own laundry.

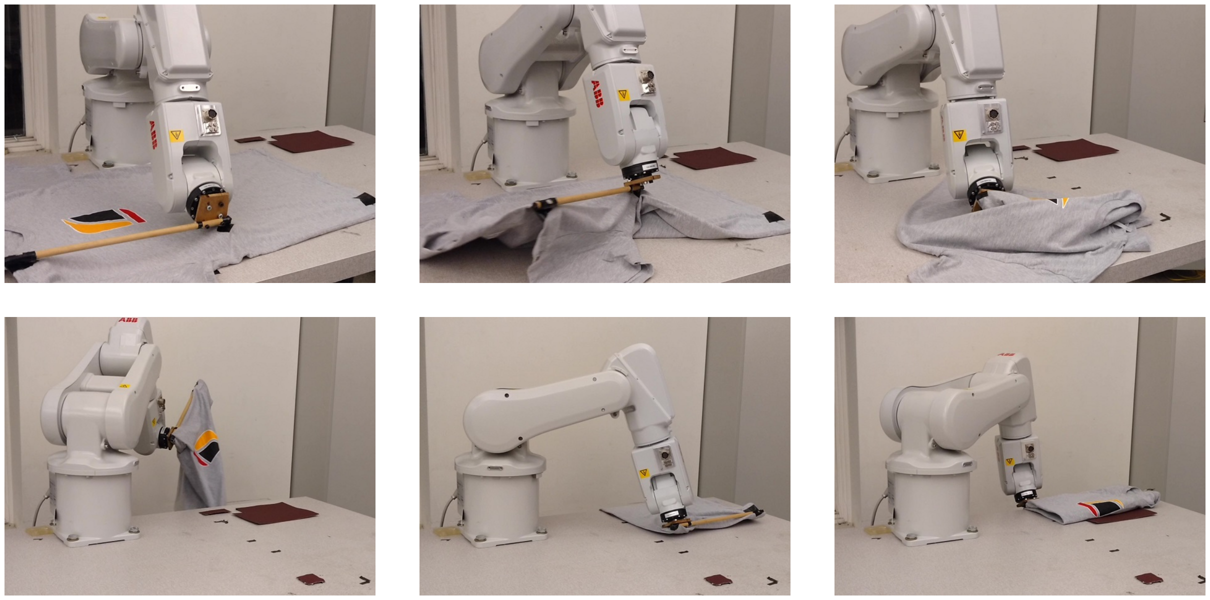


Figure 3: Our robot performing Japanese T-shirt folding

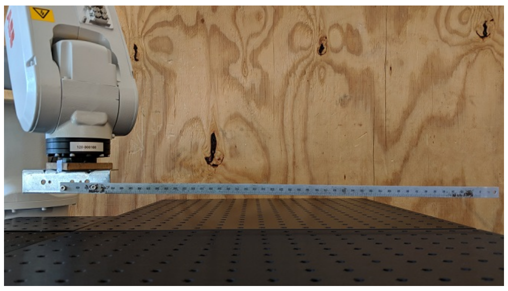


Figure 4: Our bar-shaped fixture

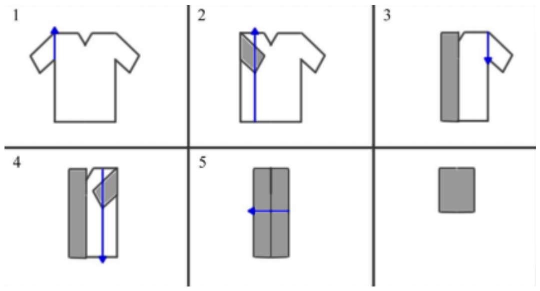


Figure 5: The fold lines used on a T-shirt [1]

4 Bar-Folding Method

We attached a long, thin metal bar to our six-jointed robotic arm, as shown in Figure 4. For each fold, the robot performs a sequence of actions: sliding the bar under the shirt through a gap in the table, which is positioned directly underneath the desired fold line; lifting the shirt until it is no longer touching the table; setting the shirt down on the table so the gap is underneath the *next* fold line; and withdrawing the bar. We use the same concept of “fold lines” as van den Berg *et al.* – the difference is that instead of immobilizing two corners or edges of the shirt with grippers and lifting the part that falls on one side of the fold line, we immobilize the shirt under the fold line itself and lift the entire garment off the table. A video is available here.

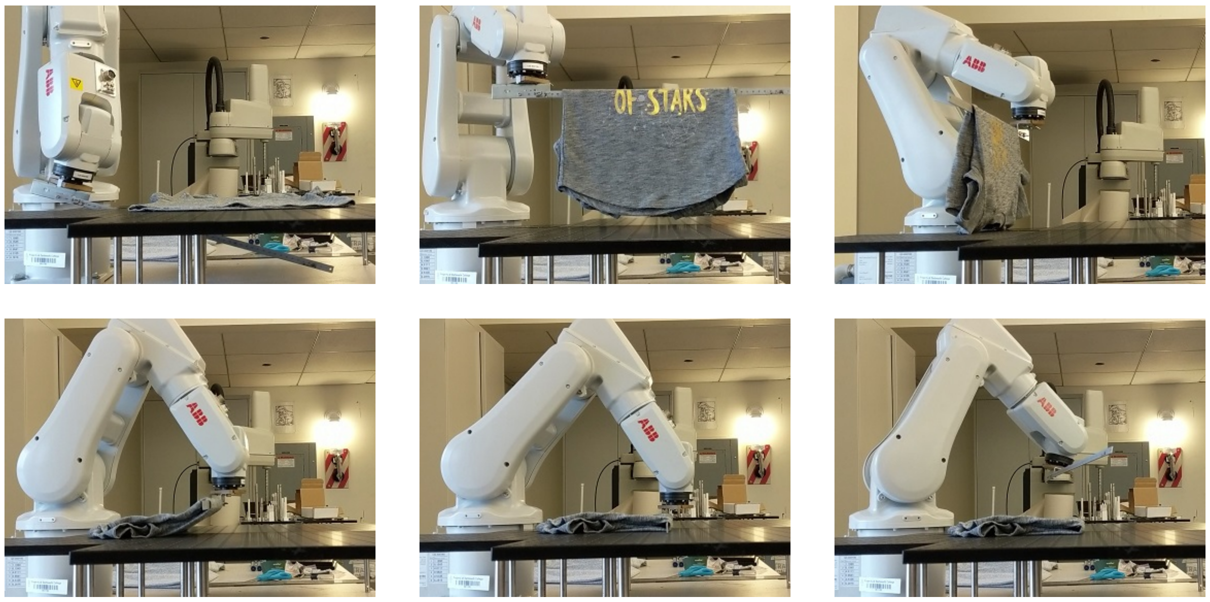


Figure 6: Our robot performing one fold using the bar-folding method

Our approach has several advantages:

- Uses just one arm, and no grippers.
- No need to identify grasp points, which requires computer vision and increases error.

- Gravity unwrinkles the shirt when it is picked up.
- Substantially faster.

However, it does have some disadvantages:

- Some folds are no longer possible, because the shirt will not be fully immobilized in the air, and may experience “buckling”.
- Some sequences of folds are no longer possible, because lifting the garment causes earlier folds to unfold.

We also observed that the process of choosing fold lines could be automated. In other studies, the researchers provide a series of folds for the robots to perform – often the same folds that people use to fold their own laundry. However, there might be other sequences of folds that achieve similar results in less time. To search for these better sequences, we perform a simulation on our simple polygonal model of the garment, testing various fold lines and considering the garment’s new shape after that fold. Section 5 describes our simulation and search strategy in more detail.

We reconsidered the end result of the folding process as well as the steps required to get there. Most robotic laundry folding strategies aim to leave the garment in a square or rectangular shape, but there may be situations where another shape is better. Our fold planner handles these situations by accepting a “target shape” – which could be a triangle or any other simple polygon – and then looking for a sequence of folds that leaves the garment in a shape as similar to the target as possible.

5 Folding model and planner

This section was written by Josiah Putman ’20, who assisted me with this part of the research; see the Acknowledgments section for more information about Josiah’s excellent contributions.

We propose a stacked-polygon model for representing the state of the folded article and an algorithm for planning folds to reach an arbitrary target shape. We then develop a planning algorithm that explores the state space of possible folds and configurations using a standard prioritizing depth-first search algorithm. Primitive actions are selected using a variety of heuristics based on common folding strategies. Individual states are evaluated using a AreaDifference function. The specific strategies and implementations of all components of the model and planner are discussed in this section.

5.1 Stacked polygon model

Instead of using a computationally intensive physical model, we develop a simplified stacked-polygon representation of the articles being folded. Each layer of clothing is represented by a polygon layer, and the entire stacked polygon is made up of a sequence of layers.

We represent individual polygons by a sequence of vertices:

$$P = (v_1, v_2, \ldots, v_n)$$

Where each $v_i \in [n] \in \mathbb{R}^{\not{k}}$.

Each layer of the stacked polygon L is a set of polygons:

$$L = \{P_1, P_2, \dots, P_n\}$$

Where each P_i, $i \in [n]$ is a polygon.

Each stacked polygon is a list of layers and a target polygon.

$$SP = (L, T)$$

This model ignores the physical behavior of cloth and assumes that perfect folds are performed at each action. It also does not keep track of where the creases between the folds are, so it treats each layer as being completely disconnected from the rest of the article.

5.2 Folding stacked polygons

Stacked polygons support the primitive actions of linear folds. For the purposes of easy translation between fold actions on the model and machine instructions, folds are represented by the following 3-tuple:

$$F = (P, \theta, \Delta x)$$

Where $P = (x, y)$ represents a rotation point, θ represents the angle of rotation for transforming the polygon, and Δx represents a horizontal shift. All folds are performed over a stationary fold line $x = x_F$. For the purposes of experimentation, x_F has been arbitrarily selected at 150 mm.

Each fold operation in the model consists of an alignment step and a folding step. during alignment, the entire stacked polygon is rotated about P by angle θ and translated in the x direction by Δx. After this transformation, the polygon is split over the line $x = x_f$, and all polygons to the right of the split line are flipped. The model normalizes the y position such that the minimum y position of any point in the stacked polygon is 0.

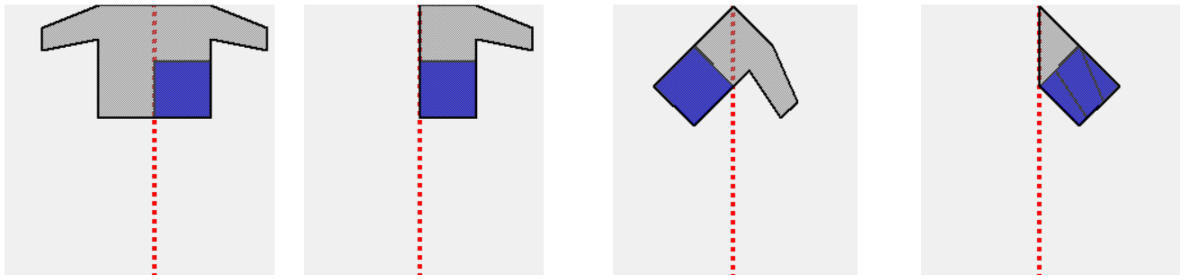


Figure 7: Successive folds on the stacked polygon

5.3 Enumerating successor states

Let $s' = T(s, f)$ represent performing a fold f on stacked polygon s.

Each stacked polygon considers a discrete set of possible folds, denoted as F. Each fold in F generated by one of the following procedures:

1. Fold line between two vertices
2. Fold line between a vertex and a midpoint
3. Fold line divides the polygon in half.

Any selected fold line $l = ((x_1, y_1), (x_2, y_2))$ can be transformed into a corresponding fold f by the following formula:

$$f = ((x_1, y_1), \theta, \Delta x)$$
$$\theta = 2\pi + \arctan \frac{y_2 - y_1}{x_2 - x_1}$$
$$\Delta x = s_f - x_1$$

The set of all successor states of s is defined by $S'(s) = \{s' : s' = T(s, f), f \in F\}$.

5.4 Prioritizing depth-first search algorithm

We use the following algorithm to explore the state space of the folding problem. Successors are produced by the process described in the previous

```
Data: stacked polygon
Result: Complete path from $s_0$ to $s_n$
$visited \rightarrow \emptyset$;
$toVisit \rightarrow stack(s_0)$;
while not exceeded number of steps do
    $s_i \leftarrow toVisit.pop()$;
    if $s_i \notin visited$ then
        if $s_i$ is goal then
            return $backtrace(s_i)$;
        end
        add $s_i$ to $visited$;
        for $successor \in sorted(successors)$ do
            push $successor$ to $toVisit$.
        end
    end
end
```

Algorithm 1: Prioritized depth-first search

5.5 Scoring function

Let $P_t = ((x_1, y_1), \ldots, (x_n, y_n))$ represent the target polygon of the folding planner and $P_s = ((x_1, y_1), \ldots, (x_n, y_n))$ represent the current silhouette of the stacked polygon being scored. Assume that the centroids of P_t and P_s are equivalent (the polygons have been aligned). For the AI to prioritize solutions that make the current stacked polygon's silhouette geometrically closer to the target shape, we are using an negative area difference scoring function. Let ϵ_{max} represent the maximum percent error allowed between the areas of the current stacked polygon and the target polygon.

Let $T = \{p = (x, y) : x, y \in \mathbb{R}, p \in P_t\}$ and $S = \{p = (x, y) : x, y \in \mathbb{R}, p \in P_s\}$. The scoring function can be defined as follows:

$$\textsc{Score}(T,S) = \begin{cases} \textsc{Area}((S-T)), & \text{if } \textsc{Area}(T-S) < \epsilon_{max} \\ -\infty, & \text{otherwise} \end{cases}$$

Because the planner does not allow unfolding, any state where the area of the target polygon cannot be imposed perfectly onto the stacked polygon is an invalid state.

5.6 Static Target Shape

An alternative approach to scoring the states of the planner is using a static target shape that is transformed along with the stacked polygon. An initial position is chosen at the start point and all fold lines must never intersect with this target shape. Area differences can be used the same as before.

6 Executing the Plan

The fold planner does not consider the details of moving the robotic arm to perform each fold. After it develops a sequence of folds, it passes each one individually to the "movement planner", which determines the best way to perform the fold given the constraints of the workspace. The movement planner receives a representation of the fold that consists of just two numbers: the angle of the fold line, and the distance from the end of the arm where the fold line intercepts the ruler. To illustrate what these parameters mean, Figure 8 shows a rectangular garment draped over the arm's extension, with a fold line, in blue, defined by its angle and intercept.

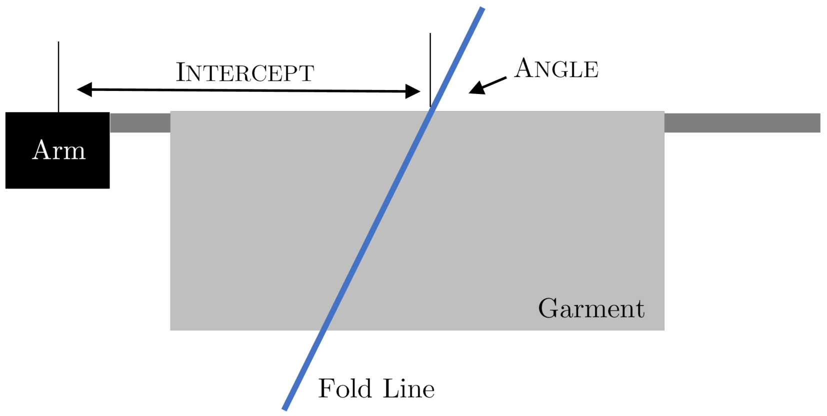
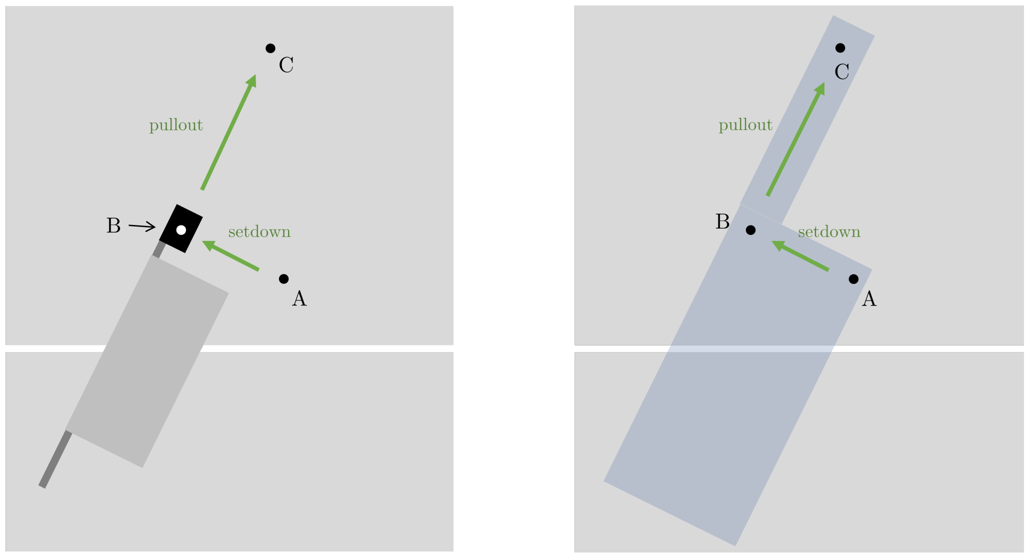


Figure 8: Explanation of Fold Parameters

Generating Laydown Orientations

After picking up the garment, the arm must set it down so that the fold line is directly over the gap in the table. This can be done in four different ways, as shown in Figure 9, with green arrows to indicate the direction of motion. After the garment is set down, the arm extension must be slid out from inside the folded fabric – an action that can be impossible to



(a) Points A, B, and C (b) Area of the Polygon

Figure 10: Collision area of laydown movement

perform if the garment is laid down at certain angles. Considering all four possible laydown orientations helps the movement planner avoid this problem.

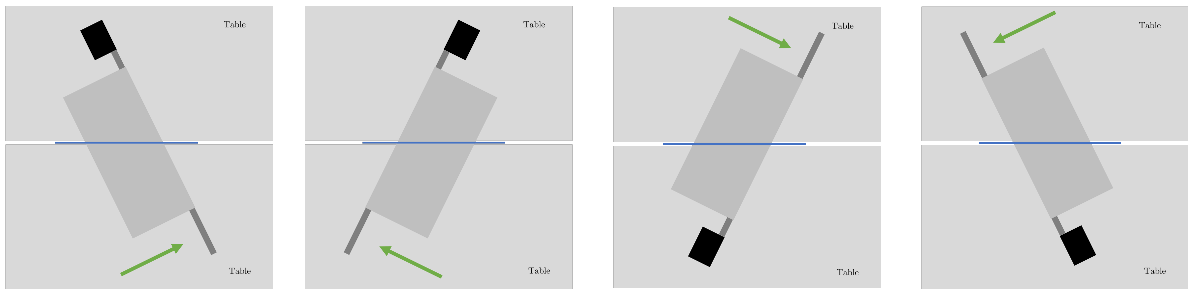


Figure 9: Four different ways to set the garment onto the table.

Comparing and Optimizing Laydown Paths

For each of the four laydown paths, we calculate the xyz location of the end effector at three points during the movement: a) before the setdown, b) after the setdown, and c) after the pull-out. Using these points, plus the angle and length of the arm attachment, we create a polygon that represents the collision area of the movement, as shown in Figure 10. The movement planner "slides" the collision-area polygon across a simulation of the workspace, as shown in Figure 11, evaluating each possibility. If any of the three points lies outside the arm's range, that possibility is thrown out. The remaining, valid options are pooled with those generated from the other three laydown orientations; finally, the movement planner considers the full set to find the most convenient one.

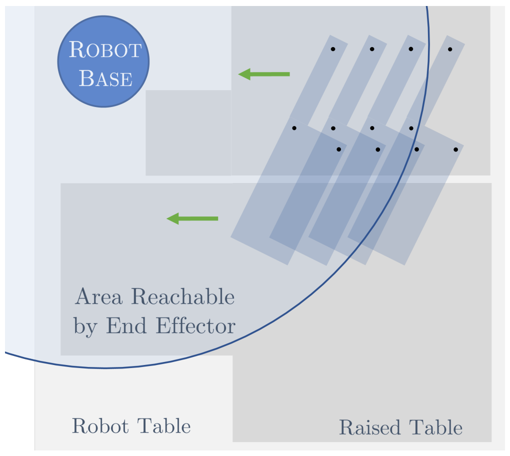


Figure 11: Considering possible laydown locations in workspace

Controlling the Robot

Once the movement planner chooses a good path for laying down the garment, it instructs the arm to assume certain (a, b) pairs, where a is an xyz-location and b is a quaternion that describes the orientation of the end effector. The reverse kinematics are handled by software on the robot's internal computer. The same controls are used for the action of sliding the bar under the garment and lifting it; this action, unlike the setdown movement, is performed the same way for each fold. The Python control interface we used is available online here.

7 Limitations and Future Work

Our work does not address the hardest part of the folding process: un-crumpling a random garment and setting it on the table in a particular place. This task requires more complex models, computer vision, and, we believe, a robot with at least two arms. Complete "pipelines" which handle this task, such as the one developed by Doumanoglou and his nine co-authors, are more comprehensive than our work. However, we believe that the bar-folding method can be useful as an addition to a larger laundry-folding pipeline. For example, a two-armed robot might un-crumple and position a shirt, then pick up a bar-like tool in one arm in order to use our folding method. This general approach – using a simple tool to perform motions that a human might not intuitively consider – could yield benefits for other stages of the laundry-folding process as well.

8 Acknowledgments

This work would not have been possible without the guidance and support of Professor Devin Balkcom, my thesis advisor, who provided the 6-dof robotic arm and also proved several important theorems related to cloth immobilization. In addition, Josiah Putman '20, a student employee of Dartmouth's robotics lab, devoted more than a hundred hours to this project as part of his job; he wrote the Python code for the fold planner, contributed a

section to this paper, and served as an invaluable sounding board for ideas throughout the process.

References

[1] Jur van den Berg, Stephen Miller, Ken Goldberg, and Pieter Abbeel. "Gravity-Based Robotic Cloth Folding." Algorithmic Foundations of Robotics, 2010.

[2] Matthew Bell. "Flexible Object Manipulation." Dartmouth Computer Science Technical Report (Ph.D. Thesis), 2010.

[3] Andreas Doumanoglou et. al. "Folding Clothes Autonomously: A Complete Pipeline." IEEE Transactions on Robotics, 2016.

[4] Matthew Bell and Devin Balkcom. "Grasping Non-stretchable Cloth Polygons." International Journal of Robotics Research, 2010.

[5] Kyoko Hamajima and Masayoshi Kakikura. "Planning Strategy for Task Untangling Laundry – Isolating Clothes from a Washed Mass." J. Robot Mechatron, 1998.

[6] Bryan Willimon, Stan Birchfield, and Ian Walker. "Classification of clothing using interactive perception." Robotics and Automation (ICRA), 2011.

[7] Yasuyo Kita and Nobuyuki Kita. "A model-driven method of estimating the state of clothes for manipulating it." Applications of Computer Vision, 2002.

[8] Yinxiao Li et al. "Regrasping and Unfolding of Garments Using Predictive Thin Shell Modeling." Robotics and Automation (ICRA), 2015.

[9] van den Berg *et al.*'s video is available at http://rll.berkeley.edu/wafr10-gfolds/.

[10] Doumanoglou *et al.*'s video is available at https://www.youtube.com/watch?v=8TsLkpPsdKo.