LaundryBot: Robotic Arms that Fold Clothes

Ben Brady
bradybt@umich.edu

Brad Frost
bfrost@umich.edu

Megan Oosthoek
oosthoek@umich.edu

Nitish Paradkar
nitishp@umich.edu

I. Project Purpose

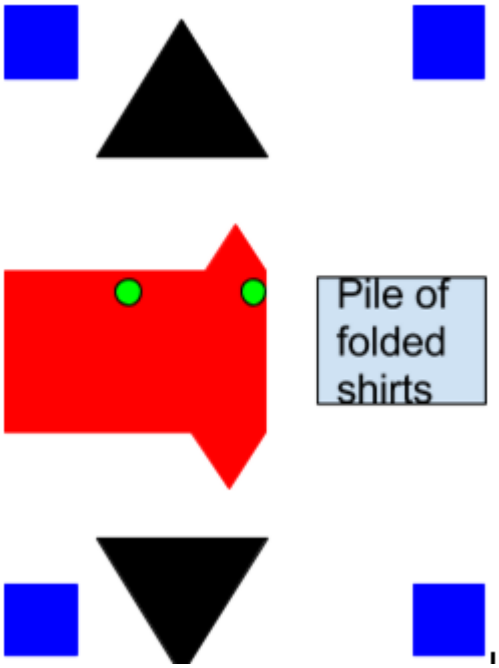

Fig. 1. The figure above shows the original setup for our environment to fold clothes. Our setup consists of two Rexarms and an overhead camera. This image shows the setup from the point of view of the camera. The blue squares represent the area of the image that is considered; the black triangles represent the positions of the two Rexarms; the red segment is the shirt that needs to be folded; the green circles represent areas of the shirt that the two Rexarms need to grab. Once the Rexarm is done folding the shirts, it will place it into the light blue box titled "Pile of folded shirts" and wait for a new piece of clothing to be inserted into its viewing area.

Our team's plan for the final project was to have a robot fold both shirts and pants. We used two Rexarms and an overhead camera. The setup of our environment can be seen in Figure 1. ideally, the Rexarms would perform the various steps in the laundry folding process. Given a well laid out piece of clothing, the system would to be able to detect whether this piece of clothing is a shirt or a pair of pants. With this info, the system would then attempt to properly fold the article of clothing, and then place it into a pile of similarly folded clothes. If we are successful, the robot should have transformed a fully laid out article of clothing into its neatly folded counterpart.

This was a worthwhile endeavor because folding and flattening clothes seems to be a challenging problem within the robotics community where there has been a moderate amount of success. Therefore, any significant advance in clothes folding would be a noteworthy accomplishment that sheds more light on the laundry folding problem. Being able to efficiently fold laundry would almost certainly improve the lives of the common man who would no longer need to worry about folding his clothes fresh out of the dryer because LaundryBot would be there to help. This would be just one step in automating the menial tasks of everyday life so that humans can spend their time on more worthy challenges than folding their clothes.

In order to fold the clothes properly, the Rexarms would follow the steps for folding a shirt as shown in Figure 2, and follow the steps shown for folding a pair of pants as shown in Figure 3.

II. Contributions to Current Research

The current state of the art research being done related to laundry folding is a team at the University of California Berkeley. Professor Pieter Abbeel leads a research team there that has been working on the laundry problem for several years now. At this point, the team has been able to get a robot to fold a basket of laundry. However, it takes them at least a couple hours to do so. They are using a robot that has very precise gripper arms specific to folding laundry. In addition, they have spent time researching state of the art computer vision algorithms focused on fine manipulation of items, specifically with the application of folding laundry.

The tools that we used that are state of the art are from the OpenCV library. This library is in wide-spread use for many computer vision applications. During development, we found that OpenCV was lacking in flexibility in several different library functions. Thus, it could be possible for us to contribute to OpenCV to improve flexibility. For example, we found it necessary to be able to use the blobs objects that were detected. The OpenCV blob detection function only gives you blob locations. It would be simple to return a reference to the list of blob objects that were detected in the library, or at least provide additional function to get more information about the blobs.

III. Plan of Execution

The laundry folding robot was split up into two major components. The arm controller component allowed us to move the end-effector of the robot arm to a point in (x, y, z) as well as control a finite state machine to determine the method to fold the clothes. The image processing detected the clothing type as well as assist in choosing the points for which the arms

Fig. 2. This image shows our method of folding a shirt. First the horizontal and vertical lines were identified as shown in the second image. The vertical line was defined as the line that cuts the shirt in such a way that there is 1/4th of the shirt on the left side of the line, and 3/4th of the shirt on the right side of it. The horizontal line was defined as the line that divides the shirt into two halves when it cuts through the shirt horizontally. Then, points A, B, and C are correctly identified as shown in the third image. A is defined as the intersection between the horizontal and the vertical line, and B and C are defined as the endpoints of the vertical line. Once these points were identified, one can grab points A and B as shown in the fourth image. Then move the arm that is grabbing point B to also pick up point C as shown in the fifth image. This shirt can then be lifted up, and the arms uncrossed as shown in the sixth image. Finally, the table can be used to perform the final fold as shown in the seventh image, to produce the perfectly folded shirt as shown in the eighth image.

Fig. 3. The following series of images show our method for folding pants. First it was assumed that the pants were well laid out as in the top left image. Then the Rexarms grabbed the ends of the pants, and moved them to the top as shown in the top right image. Then, the pair of pants were rotated 90 degrees as shown in the bottom center image. Finally, one final fold was performed by grabbing the top side of the pants and moving them to the bottom, thus resulting in a pair of well folded pants as shown in the bottom right figure.

Fig. 4. The following pictures show our method for grabbing areas of a well laid out clothing component. The arm started with the claw open, and touching the floor as shown in the right image. Then it moved forward (with the claw touching the floor), so that it could create a bigger section of clothing to grab, and finally closed the claw as shown in the left image to securely grab the piece of clothing.

could begin folding. The two components will be described in detail below.

Arm Control

In order for the Rexarms to be able to fold both shirts and pants, they need to follow the steps show in Figures 2 and 3. To perform these steps, feedback from our image processing component was used to command the Rexarms where to go. In order to help make the Rexarms more maneuverable and reach arbitrary points more easily accessible, several new models to help achieve this goal were introduced. These various models can be seen in Figures 5, 6 and 7. With the help of these models, it was ensured that the end effector of the arm could be both close, far, and high up from the base of the robot arm. This helped allow the arm to grab more areas of clothing easily, and also allowed it to help avoid collisions with the other arm.

Due to the limited ability of the claw, the first major problems we had to deal with while working on this project was being able to correctly grab pieces of clothing. We resolved to gliding the end of the claw against the fabric in order to create a ”clump” that could easily be grabbed whenever the claw was closed. An example of this procedure can be seen in Figure 4. this method of grabbing was decided on because we could not think of many easy ways to modify the claw such that it consistently and reliably grabbed the piece of clothing. This method was found to work significantly better than applying a bar across the claw (the purpose of this would be so that it provided uniform pressure across the piece of clothing).

Another major problem was finding a new way to fold pants. The original proposed method of folding pants looked like something similar to Figure 8. However, it was quickly

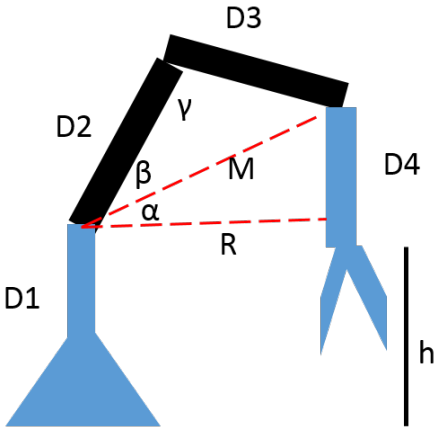

Fig. 5. The model used to reach short, low points. It is the model explained in lecture and used in A3. This model requires that D1 and D4 are parallel and both perpendicular to the ground. We define h as the distance from the top of the claw to the ground, and R as the horizontal distance from the end of the claw from the base of the robot. We use the variables α, β, γ, and M to help calculate the angles to set each joint of the Rexarm to so it can correctly follow this model.

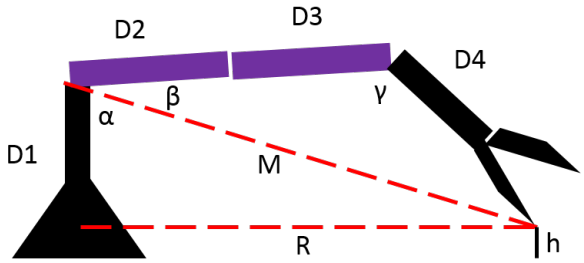

Fig. 6. The model used to reach far, low points. This model requires that D2 and D3 are at a 180 degree angle. We define h as the height we want the end of the claw to be off the ground. This will mostly be a very low value. We also define R as the distance away from the base that we want the end effector of the claw to be. We use the variables α, β, M, and γ to help find the values to set the joints of the Rexarm to in order to follow this model.

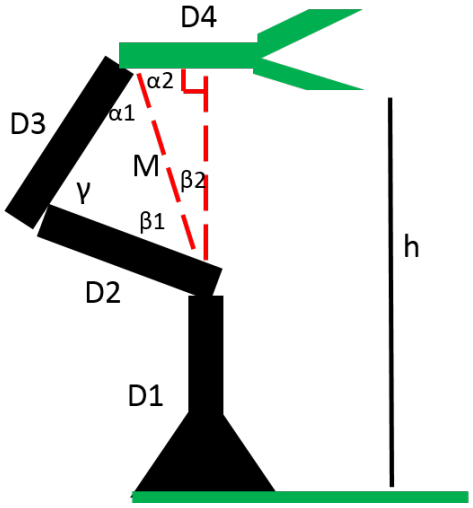

Fig. 7. The model used to reach short, high points. This model requires that D4 is parallel to the ground. We define h as the height we want the claw to be, and we similarly use the variables M, γ, α_1, α_2, β_1, and β_2 to help calculate the correct values to set the joints in the Rexarms to correctly follow this model.

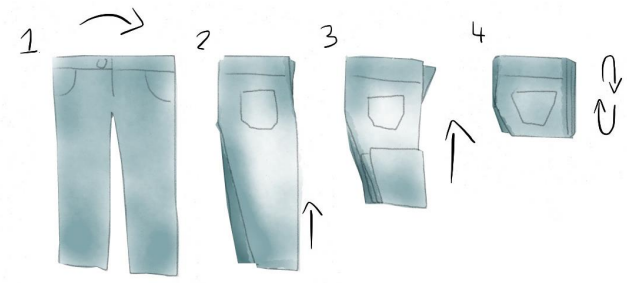

Fig. 8. The proposed method of folding pants. In the first step, one would grip the top two corners of the pants and fold one pant leg over the other. Next, the bottom two corners of the pants are grabbed and folded half way up the pant leg. Finally, the bottom two corners are grabbed again and folded up to the top two corners of the pants.

realized that the Rexarms could not grab both the bottom and top of the pants simultaneously to create a fold as shown from the first image to the second image in Figure 8. The method as shown in Figure 3 was used instead. This allowed the Rexarms to easily grab both the top and bottom of one side of the pants to create a final fold as shown in the bottom right image of Figure 3.

Another issue was the Rexarms colliding together. In order to solve this problem, synchronization code was added. This was very helpful because the Rexarms often needed to do things in parallel. This was done by running each arm controller in a separate thread. However, the Rexarms sometimes took different routes to complete a path. These offsets in synchronization were often not deterministic simply due to the nature of multi-threaded programs. In order to help fix this problem, both a wait() and signal() method were added for each arm controller, so that it would wait for the other arm to signal that it was ready to proceed. The wait() and signal() functions simply used a global mutex and bool to help check if each arm was ready.

In order to help us while testing, a CollisionDetector class was written that monitored the state of the joints for the two Rexarms. Using the position of the joints, forward kinematics was used to help find the endpoint of each claw. This gave us the location of the claw for each arm in its own frame of reference. To help see if these points were too close to each other, one of the points was transformed to be in the frame of reference of the other arm. This was a simple transformation as both arms had more or less the same coordinate frame, except the positive x-axis for each arm was pointing in the opposite direction. Thus, we only needed to change the x value for the point, and this was simply done by the following equation:

$$x_{new} = DISTANCE - x_{old}$$

. The variable $DISTANCE$ in this equation represents the distance between the two arms.

Image Processing

For the Rexarm to know which points to grab, it asked the image processing component via an LCM message for the points it should begin folding at. This means that the Rexarms were agnostic to the folding type until the image processing component told them the type. When this request was initiated, the image processing component began running for several iterations. The pipeline of each iteration is described below.

1) Grab frame from camera and crop into usable rectangle.
2) Cluster the image using K-Means into four different colors: robot arm (black), registration squares (orange), background (white), and the clothing item color.
3) Perform blob detection on the clusters to further segment items in the field of view into usable objects.
4) Filter clothing item blob out from the rest of the blobs.
5) Determine size of blob to detect clothing item (shirts always larger than pants in our case).
6) Gain useful information about clothing item blob such as height and center to determine points to grab.
7) Convert pixel coordinate points into arm coordinate points using a similarity transformation.

To grab a frame from the camera, the `ImageSource` library was used that is available to all EECS 467 students. With this library, we are able to adjust camera settings and easily retrieve an allocated pointer that represents an array of pixels in the format we specify. For this project, it was necessary to use RGB format so that we could cluster by color. The image was then displayed using Vx, and had a clicking mechanism to choose points on the image to crop. This allowed us to create a rectangle that we could crop from each time a frame was received from the camera. It was necessary to crop the image to easily cluster a smaller area of the in the field of view of the camera that we cared about. Otherwise, an unnecessary amount of computation would be added for the clustering and blob detection algorithm.

After the image has been cropped, the K-means clustering algorithm was used with a k-value of 4 to determine which cluster each of the pixels in the raw image should belong to. This operated by finding the four colors c_i such that the following expression is minimized:

$$\sum_{i=1}^{4} \sum_{x \in S_i} ||x - c_i||$$

where S_i is the set of pixel colors whose closest mean value is c_i. Then all of the pixels were iterated through and assign their new color to be C_i. Because registration squares were being used to create the image transformation, there was some difficulty in picking the correct color for the registration squares. If too dark a color was used for the squares, they would be clustered together with the pants, and this would make it harder to pick out the registration square blobs. It could probably still be done by examining blob size and just picking the smaller blobs as the registration squares, but there would still be some problems because the shadows on the image tend to get clustered into the shade of blue. If the color of the registration squares were too bright, it would get clustered with the white, so we finally settled on using a shade of orange for the registration squares because it didn't conflict with the arms, shadows, or any piece of clothing that we were folding.

After the K-means had been completed, depth-first search was used to find blobs in the image using the K-means cluster colors. This works by selecting one pixel that has not been assigned to a blob yet and then iterating across the 3 to 8 pixels that border it. During the iteration across these boundary pixels, we check to see if they are already assigned to a blob. If they were, they were skip. Otherwise, if the pixel was clustered into the same color as the center pixel, then they were added to the center pixel's blob and added to a stack of pixels to repeat this process on. If the pixel was not clustered with the center pixel's color, then it is not added to the blob. This was performed until all of the pixels belonged to a blob.

The cases that there are some very small blobs in seemingly random positions or within other blobs such as the shirt (the logo) or the arm (reflections from the metal pieces) that are undesirable and unnecessary to analyze. Therefore, the blobs that fall beneath a certain area threshold were filtered out. This was tuned using a function called `cleanBlobs()`. A good value for this threshold was around 3 percent of the total area of the cropped image. Here, the area of a blob is just the number of pixels that belong to that blob.

In order to find the points we needed to pick up to do the fold, the blob that was the piece of clothing needed to be identified. this was done by finding the blob that was closest to the center. That is, the center of the image was computed to be

$$\left(\frac{imageWidth}{2}, \frac{imageHeight}{2} \right)$$

were the units are in pixels. Then, the centers of each of the blobs was computed by averaging the x and y values over all pixels within the blob. Therefore the center of blob B is (x, y) where

$$x = \frac{1}{|B|} \sum_{(x_i, y_i) \in B} x_i$$

and

$$y = \frac{1}{|B|} \sum_{(x_i, y_i) \in B} y_i$$

The distance between the center of the image and the center of the blob is then just the standard Euclidean distance in the units of pixels. This method was not completely robust to begin with because sometimes the background white blob would be recognized as the center blob instead of the clothing, so this was fixed by ignoring the center of white blobs. White blobs were defined to be blobs that had a saturation value below .1 which needed to be tuned. This fixed one problem, but sometimes a shadow around the pants would get recognized as the blob closes to the center. A good solution to

this issue was not found; however, it could probably be fixed by ignoring blobs with a small area to perimeter ratio.

After the clothing item blob was filtered out from the rest of the blob collection, we now have a useful object that we can perform computations on. All of the blobs were represented in a `Blob` class that had several useful helper functions such as `Blob::getCenter()` and `Blob::getHeight()`. For example, the `Blob::getCenter()` function found the means x and y values of all of the pixel coordinates that were members of the particular blob. In addition to these functions, a `Blob::getSize()` function was implemented that just kept count of how many pixels were in the blob. Using this, a clear boundary that separated the shirts from the pants was determined and allowed us to classify the clothing item. We had originally attempted to classify the clothing items via corner detection, but found that the OpenCV algorithm we were using was too unstable on a live image for this to be a suitable classification.

As mentioned before, we implemented functions to get the center and height of a blob. With this information, a set of heuristics was used to determine the points on the clothing blob to send to the Rexarms as grabbing points. For the shirt, the center was found and then calculated a percentage of the width of where to grab along the middle for one of the arms. Then, from this offset in the x direction along the middle, another offset was calculated in opposing y directions towards the top and bottom of the shirt. These points were used for the other arm. Figure 9 visualizes how these points were chosen.

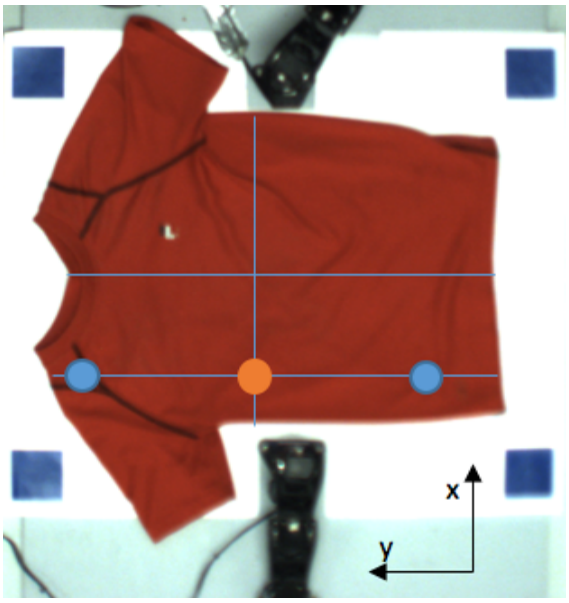

Fig. 9. Image representing how the shirt points were determined. The center point of the blob would be determined, from which the first point (orange) along the x axis of the center of the shirt blob could be determined. Then we move a specific offset in opposing directions along the y axis of the first point we chose. These are the blue points.

For the pants, similar calculations were done to that of the shirt. However, this had to be performed for several different states. the height, width, and center of the blobs were used to get the points we desired. Figure 10 shows how the points were determined for a pair of pants.

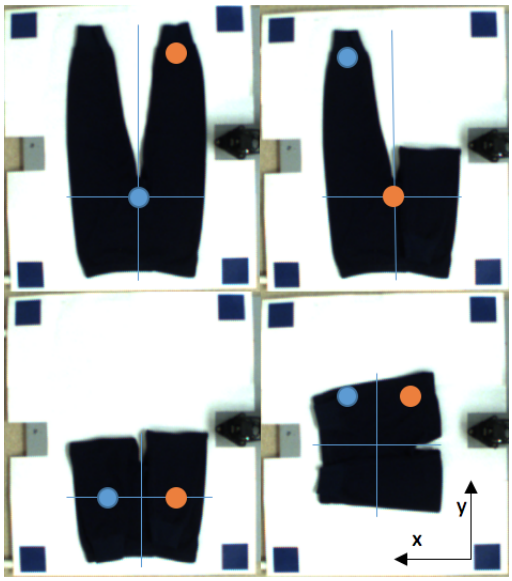

Fig. 10. This image shows the methods we used to determine the points for the arm to grab. The lines represent the height and width of the pants along the center line, which were usable features of the clothing blob object. In the top left corner image, the middle of the pants were given to one arm to hold done while the other arm went long the height of the pants in the middle and then some offset of the width of the pants to the right to grab the pant leg. Then it could fold the pant leg up. In the top right image, the same treatment was done to the other pant leg, but with arms swapped. In the bottom left image, the grab points were assigned by finding the center and moving along the x axis in opposing directions at some fraction of the width. This allowed The Rexarms to grab points to slide the pants towards the middle of the folding area. In the bottom right image, two points along the top edge of the pants were determined to grab and fold in half.

To implement this, a base class called `ClothingItem` was implemented with a pure virtual method called `ClothingItem::separatePoints()`. A `Pant` and `Shirt` class were derived from the `ClothingItem` class and they each implemented the `ClothingItem::separatePoints()` separately. In this method, the pant points and shirt points were chosen, depending on which type of clothing item it was. At first, corner detection was used to determine points in these classes. But as mentioned previously, we found the algorithm to be too unstable for finding usable points to grab.

Now that the points have been determined for the Rexarm to grab in pixel coordinates, the points were converted into Rexarm coordinates by using a similarity transform. A transformation was performed for each arm, since they had mirrored frames of reference. To do this, the angle that each arm was turned relative to how the camera viewed the pants was determined. Then, the translation that the base of the arm was relative to the bottom left corner of the image was determined. Finally, a scale factor was determined by measuring the ratio of the distance between registration squares in metric units and the distance in pixel units. Figure 11 below shows each arms' frame of reference.

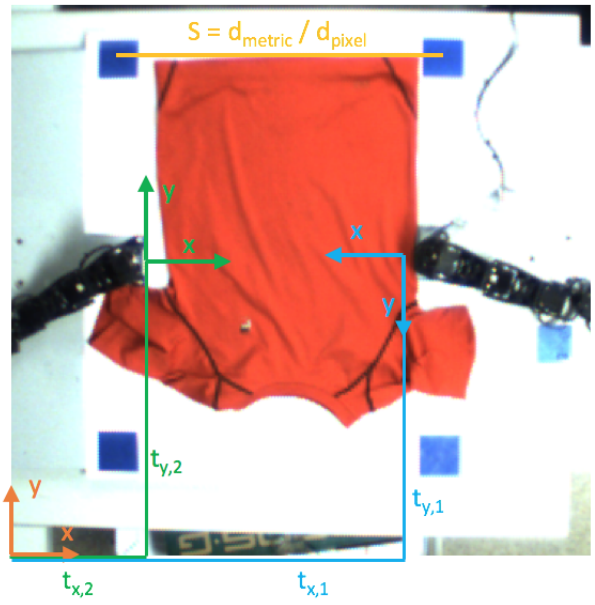

Fig. 11. This image shows the important frames of reference that are used for converting a point in the camera's frame of reference to a point in each arms' frame of reference. The orange coordinate reference in the bottom left image represents the camera's frame of reference, the blue represents Arm 1's frame of reference, and the green represents Arm 2's frame of reference. As we can see, both the camera and Arm 2 share the same orientation, thus no rotation needs to be applied for that transformation. However, the blue needs to be rotated by π radians. The frames of reference are translated by (t_x, t_y), relative for each arm to the camera frame of reference, as shown in the image. In addition, the pixel values are scaled by the scale factor shown near the top of the image, which was the ratio of the distance between blue squares from metric value to pixel value distance.

IV. Lessons Learned

Arm Control

Though eventually collision detection was written, it would have been more helpful and better for the servos if it were written earlier in the process. It would have especially helped with writing the synchronization code between the vision and arm. It caused a great deal of stress on the servos and there were times that the servos were so overheated that they would instantly fail. Usually after a good night's rest the servos would be up and running again but a couple of times the servo had to be replaced. This took a lot of time and effort away from coding.

Image Processing

One thing that would have been better was to use something else instead of registration squares for the image transformation. Because the processing relied on the squares being a particular color that was non-existent elsewhere on the field, it was challenging trying to find the right color to work with our clothes and avoid shadows. It may have been better to use AprilTags so that we could more easily find the registration squares and not worry about their color. This would have saved time thinking about the right color and how it would interact with the arms and clothes.

Even if AprilTags were not used, there should have been a method to automatically identify the color of the registration squares so that we didn't need to manually change the color every time we tested a new registration square color. Our team didn't have a good way of finding this color automatically, but this may have been a function of our team not finding the time to dedicate to solving that problem.

During the development of the image processing, a lot of time was spent on how to detect the points to grab. After doing some research, it was found that using corners would be a good feature to detect points to grab. This seemed intuitive because corners were meaningful features on a piece of clothing. As long as it was possible to confidently detect what type of clothing it was, the corners could be used as a starting point for the arms to grab. Several specific search functions were implemented to find the corners that were desired, such as finding the top right-most corner. However, the OpenCV corner detection algorithm was very poor at finding stable corners. The number of corners would fluctuate between 3 and 20 consistently, and most of the time they would never stay the same unless it was a really solid corner. In addition, the algorithm would detect items that really weren't good corners when there were clearly better corners available. If we had written our own algorithm, this would have been a good approach, but it was difficult to try to tweak the OpenCV algorithm to work in our favor.

V. Results

Fig. 12. A folded shirt. Results varied and this was considered a 'good' fold.

A piece of clothing was considered to be folded if the Rexarms were able to grab the points and execute its motions correctly. In the end, the Rexarms were able to consistently fold shirts. We average about a 90 percent success rate. But, we did not have as much luck with the pants. In all the attempts, the pants were only successfully folded using vision twice. With the pants, folding in both of the legs as shown in the third panel in Figure 3 was consistent but, moving the pants up and rotating them 90 degrees was troublesome.

The final results of successful folds are shown in Figure 12 and 13. Though these are considered successful folds, they

Fig. 13. Folded pants. Pants were only folded with vision successfully twice. Results varied and this was the best.

are not good enough to put away in a closet. In order to make these look better, a flattening method would need to be thought of.

VI. Certification

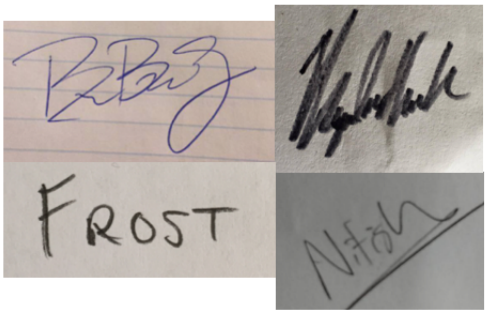

Fig. 14. The team participated and contributed to team discussions on each problem, and the team attests to the integrity of each solution. Our team met as a group on several dates over the course of the project.