The 2010 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 18-22, 2010, Taipei, Taiwan

Motion Planning for Cooperative Manipulators Folding Flexible Planar Objects

Benjamin Balaguer and Stefano Carpin

Abstract— **In this paper we consider the problem of folding deformable objects like cloth or towels. Embracing a simple object model, we present a new algorithm capable of generating collision-free folding motions for two cooperating manipulators. The algorithm encompasses the essential properties of manipulator-independence, parameterized fold quality, and speed. Numerous experiments executed on a real and simulated dual-manipulator robotic torso demonstrate the effectiveness of the presented method.**

I. Introduction and Problem Definition

The ability to manipulate flexible objects like cloth, paper and the like is a much needed addition in order to produce really reliable robotic assistants for a variety of tasks, either in domestic or industrial applications. Notwithstanding, most research in robotic manipulation assumes to deal with rigid bodies. In this paper, we take the first steps in overcoming what we believe to be some of the most limiting issues towards the development of cooperative manipulators jointly handling highly deformable objects. Opening letters, bags, boxes, or wrapping paper, playing cards, sorting papers, journals, magazines, or books, arranging clothes, bedding, or linens - all are examples of everyday-tasks involving highly deformable objects that a service robot might be expected to perform. The method we propose is based on the explicit solution of inverse kinematics (IK) to produce coordinated smooth trajectories that comply with manipulator constraints and also do not impose excessive stress on the object being manipulated. Because of these reasons, we formulated the IK based approach instead of relying on popular randomized techniques that tend to produce zigzagged paths that may damage the flexible objects being manipulated.

The paper is organized as follows. We start by reviewing our problem definition in section I-A, highlighting any assumptions made. In section II, we describe previous work associated with the modeling of deformable objects, the application of folding in robotics, and motion planning for dual manipulators. Section III covers the details of our motion planning algorithm, followed, in section IV, by experiments performed both in simulation and on a real robot. Concluding remarks, extensions, and future work are found in section V.

A. Problem Definition

We solve the motion planning problem for folding tasks of planar deformable objects (e.g. napkin, towels). Since there are many different ways cloth can be folded and scarce former research, we impose specific constraints and use this section to highlight our choices. Service robotics requires a single platform capable of completing many different tasks. As such, highly specialized robots built for a specific task should be replaced by a single, more general, counterpart. Consequently, our algorithm generalizes to any manipulator of 6 or more degrees-of-freedom (DOF) capable of performing pinch grasps. We work with rectangular planar objects and assume that we know their lengths, widths, rotation, and one corner point. Both assumptions are valid since most, if not all, towels and napkins are rectangular and capturing their lengths, widths, and corner points using vision is a straightforward process. Our algorithm specifically solves the problem of a symmetric fold, where half of the object is repeatedly folded on top of the other. Finally, since we could not find previous work on deformable object grasp planers and are focusing on motion planning, we do not attempt to physically grasp the object and assume it is possible with a pinch grasp, in a similar fashion to human grasping [6]. The actual pinching strategy is the subject of future research.

School of Engineering, University of California, Merced, CA, USA, {bbalaguer,scarpin}@ucmerced.edu

II. Related Work

A detailed review of deformable object models is beyond the scope of this paper and we briefly mention the most popular models. Interested readers are pointed to the survey in [9] for additional information. Free Form Deformations [13] can be powerful for 3D objects, but are not suitable for cloth since they do not take into account the structural properties of the object. Mass-spring systems [12] suffer from the stiffness problem [3], which occurs when integration time steps are too big and results in poor fidelity. Finite Element Methods [17] are computationally expensive and work best with small deformations. In robotics, representations of deformable objects have taken a different direction. The most popular one, only utilized for folds, decomposes a deformable object into a set of kinematic links composed of a face and a foldable edge. This representation has successfully been exploited to fold paper [14] and carton [11]. The faces are assumed to be rigid and the foldable edges are known a priori. Being simple, fast, and proven for folding in robotics, we use a variant of this geometrical model.

Song et al. look at the problem of folding paper craft in [14]. The authors formulate their work as a motion planning problem whith the object being decomposed into links and foldable edges. The formulation allows the usage of a PRM to solve the folding problem, where the Configuration space (C-space) encompasses each edge. Unfortunately the folding process does not take into account an actuating robot. A similar paper, using the same kinematic description,

978-1-4244-6676-4/10/$25.00 ©2010 IEEE

is presented in [11] for carton folding. The work differs, however, in that the authors use a tree instead of a PRM, are looking to find all foldable solutions, and implement their method on a real robot. At run time, however, an operator chooses the best fold sequence for the robot. Better robot frameworks exist for origami [2] and T-Shirt [4] folding. Both papers offer robot-dependent solutions, rendering the work useful in specialized environments but unsuitable for service robotics.

Research on dual manipulator motion planning is more readily available. In [16], Vahrenkamp et al. use two manipulators in re-grasping tasks. Their solution is RRT-based and the authors address the high DOFs of a dual-arm robot by using a randomized IK solver to analytically solve 6 DOF of the arms given randomly sampled values for the remaining joints. They show the IK solver performs better than a Jacobian-based solver. In a similar work [15], Tsai et al. look at dual-arm manipulation using RRTs to plan paths in dynamic environments comprised of moving objects. A RRT-variant is formed, adding time and cost information to dictate where the tree should grow and reduce redundant twists and turns. Gharbi et al. also look at the planning problem for dual-manipulators using a PRM-inspired technique [8]. More specifically, they consider multi-manipulators for which a PRM that takes into account the entire system will result in slow performance due to the high number of DOFs. Consequently, the authors decompose a multi-arm system into sub-components that are exploited to increase the speed of path planning.

III. Motion Planning

A. Trajectory Generation in Configuration-Space

Generating a trajectory for each manipulator is intimately tied to the model used for the deformable objects. As mentioned in the previous section, we choose a simple geometrical approach specifically invented for folding applications where a fold is represented as a kinematic chain comprised of a joint (i.e. the folding crease) connected to two rigid links (i.e. the folding faces). As opposed to the previous works [14], [11], [10], we do not assume we know the configuration apriori, allowing our algorithm to fold on top of a previous fold (a limitation of the aforementioned works). Given the knowledge of the length S, width W, object angle A, and right-most corner point $R = Rx, Ry, Rz$ in global Cartesian coordinates, we can produce mathematical equations for the two trajectories in global Cartesian coordinates. We uniformly sample data points from the trajectory by using a variable V, which ranges from 180 to 0 degrees with a defined step size. We have chosen 5 degrees as our step size and have found it to be a good tradeoff between speed and the density of data points. The geometrical process, highlighted in Figure 1, is governed by the equations below for the right trajectory. The equations can be used for the left trajectory, by generating a new point $P = Px, Py, Pz$ and substituting it for R. The new point P is computed with $Px = -W\sin(A) + Rx$, $Py = W\cos(A) + Ry$, and $Pz = Rz$.

$$
\begin{aligned}
X &= \frac{S}{2}\cos(V)\cos(A) + \frac{S}{2}\cos(A) + Rx \\
Y &= \frac{S}{2}\cos(V)\sin(A) + \frac{S}{2}\sin(A) + Ry \\
Z &= \frac{S}{2}\sin(V) + Rz
\end{aligned}
$$

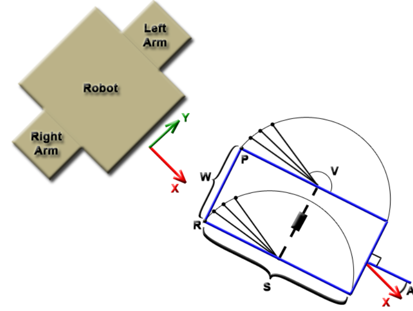

Fig. 1. Geometrical diagram used to derive the trajectories in Cartesian space, showing each mathematical variable.

The data points generated in Cartesian coordinates need to be converted to the robot's C-space. Let $L = [L_1 L_2 \cdots L_N]$ be the set of data points for one trajectory, with $L_i \in \mathbb{R}^3$. The conversion from Cartesian to C-space is achieved by using an IK solver. Even though any manipulator IK solver will work, we briefly describe the one we use with our platform, i.e. two Barrett Arms and Hands (see Figure 3). More specifically, each arm is anthropomorphic with a redundant DOF and a spherical wrist. Given a wrist orientation, we solve for IK analytically by treating the redundant joint as a free parameter. Changing the free parameter effectively allows the sampling of configurations for a given data point in Cartesian space (i.e. a given L_i). Rather than randomly sampling [16], we sample uniformly from the redundant joint's limits with a step size of 4 degrees. Evidently, the step size involves a tradeoff between speed and the density of solutions. The hand is composed of three fingers, two of which are used for the pinch grasp. The wrist orientation is constant, constrained to be parallel to the table with the unused finger pointing away from the other robotic arm. Finally, the IK solutions in C-space are pruned by removing any configurations outside the manipulator's joint limits. Put differently, each data point in L is replaced by a set of manipulator configurations C.

$$
C = \begin{bmatrix} C_{1,1} & C_{2,1} & \cdots & C_{N,1} \\ C_{1,2} & C_{2,2} & \cdots & C_{N,2} \\ \vdots & \vdots & \ddots & \vdots \\ C_{1,A} & C_{2,B} & \cdots & C_{N,C} \end{bmatrix}
$$

Note that $C_{i,j} \in \mathbb{R}^{DOF}$, where DOF is the manipulator's DOF, and that the notation $C_{i,j}(k)$ refers to the k-th joint value of configuration $C_{i,j}$. With our deformable object model, we implicitly impose two constraints on the arms' trajectories. First, the distance between the two grasping points will remain the same throughout the motion. Second, at any point on the trajectory, the relative height between the two contact points will equal zero.

B. Graph/Roadmap Creation

Having generated a set of robot configurations for each data point in the trajectory, we now focus on building a graph to be used for motion planning. We take a similar approach to [8] by generating two separate graphs, one for each manipulator, as opposed to generating a single graph representing both manipulators. Each vertex of the graph represents a robot configuration and each edge represents a connection (e.g. path) from one configuration to another. We introduce the notion of levels, where each level of the graph corresponds to a distinct global Cartesian coordinate, L_i. In other words, each level is comprised of numerous vertices, all representing the same Cartesian coordinate. Vertices are connected such that each vertex at any given level is connected to all the vertices of the next level. A path that follows the trajectory can then be generated by moving from one level to the next (i.e. moving from one trajectory data point to the next). In an attempt to make path selection easier, an initial vertex, C_I, is added as the first level of the graph and a final vertex, C_F, is added as the last level of the graph. A graphical representation of one roadmap is shown in Figure 2.

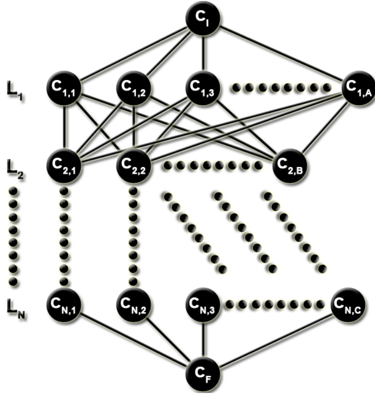

Fig. 2. Graphical representation of a roadmap.

Since we want to find the best path within this highly interconnected graph, we associate weight functions to edges. More specifically, we have defined time, boundary singularity, and collision weights as possible representations for what would describe a desirable path. The collision weight, which requires calls to a collision detector, is implemented as part of the path selection process (next section) in order to minimize the number of calls to the collision detector. Consequently, the edge cost is represented as $E_c = \alpha \times W_t + \beta \times W_b$, with $\alpha + \beta = 1$ and where W_t and W_b represent the time and boundary singularity weights, respectively. The idea behind this cost calculation is that users can scale the weights based on what their definition of what a best path is. For example, for a fast execution time α would be increased, whereas for a safe execution, β would be increased. To compute the time weight, we assume that the joints rotate at a constant velocity. This reduces the problem to minimizing the amount of rotations exerted on the joints. More specifically, given an edge between two configurations, $C_{x,y}$ and $C_{x+1,z}$, we are interested in finding the joint with the maximum amount of rotational change, which will take the longest to rotate into position. The maximum difference between the positive and negative joint limits, labeled pL and nL respectively, is used to scale the weight between 0 and 1.

$$W_t = \frac{\max_i[C_{x,y}(i) - C_{x+1,z}(i)]}{\max_i[pL(i) - nL(i)]}$$

When α is set relatively high, the time weight induces an interesting behavior. Since the best paths will have the minimum rotation amount, they will stay very close to the starting configuration, C_I. As such, our choice for C_I will dictate the type of motion executed. In addition to reducing execution time, the time weight now becomes a powerful tool. For the boundary singularity weight, we want to severely penalize rotations that are close to the joint limits for each joint. Consequently, we use an exponential function, which we scale to be between 0 and 1, where $M = pL(i) - \frac{pL(i)-nL(i)}{2}$.

$$W_b = \sum_{i=1}^{DOF} \frac{e^{|C_{x+1,z}(i)-M|} - 1}{DOF \times (e^{(pL(i)-M)} - 1)}$$

The boundary singularity weight is used for safe execution and to steer the manipulator away from boundary singularities. Moreover, and similarly to the time weight, an implicit characteristic of the weight can be deduced. Joints with limits ranging from -180 to 180 degrees can generate invalid paths as they approach one of the limits. Indeed, and as an example, as the joint approaches -180 it will, at some point, switch over to 180, resulting in a 360 degree rotation of the joint. This phenomenon is undesirable, especially if the manipulator is holding the deformable object. This weight helps the manipulators stay away from these configurations.

C. Path Selection

Conversely to [8], we find the best paths in each roadmap and merge the paths rather than merging the two roadmaps. This procedure limits the number of calls to a collision detector, an important feature of our algorithm, especially considering the high connectivity between the vertices. More specifically, we have two weighted graphs and wish to find the best combination of collision-free paths. To that effect, we start by finding the best t paths of the first graph and the best u paths of the second graph. The paths are found by running Dijkstra's shortest path algorithm on the graph starting with the initial configuration C_I and goal configuration C_F. Every time a path is found, we remove all the path's edges from the graph and repeat the process until no more paths are found (i.e. until two levels are completely disconnected from each other). While we acknowledge that there are different ways of pruning the graph to generate the paths, we have found this technique to be quite successful in finding paths that are different from each other. Simply removing one, or a few, edges would produce paths that are too similar to each other. Once again, we have a tradeoff between speed and solution density but we have found that our pruning method provides a good balance between the two, generating

anywhere between 20 and 50 paths per graph, depending on the object's parameters. Running the search on each graph results in two sets of paths, the permutation of which gives the total number of best solutions. We then propose two methods to choose a path from this set. In the first, the set of solutions is ranked by ascending costs and calls to a collision detector determines the first collision-free path, which is, evidently, the best one. In the second method, we include a new weight, W_c, which takes into account the distance between the two manipulators. Calls to the collision detector are made, returning the minimum distance, d, between the two manipulators. Since we want to penalize close manipulators more heavily, we use an exponential function. We introduce two new variables, $dRate$ and $Margin$, that dictate the rate of descent of the exponential function and the safety margin, respectively. As a reference, we used 0.8 as the rate of descent and 0.1 (i.e. 10 centimeters) as the safety margin.

$$W_c = exp\left(\frac{-d}{dRate \times Margin}\right)$$

The weight, W_c, is calculated for each pair of configurations in the path and can be incorporated into the cost function by multiplying by a factor γ and adding it to the total path cost. Effectively, this means that our new cost function becomes $E_c = \alpha \times W_t + \beta \times W_b + \gamma \times W_c$, with $\alpha + \beta + \gamma = 1$. Once the costs have been updated, the paths can be sorted in ascending order and the best one is selected. As will be shown in the experiments, the collision weight provides little improvement and suffers from a huge performance hit due to distance calculations by the collision detector. Consequently, we recommend using the first method described.

IV. Experimental Results

We performed a series of experiments on our robotic platform, a static torso composed of two Barrett Arms and Hands, both in simulation and on the real system (see Figure 3). For the experiments that we present in this section, we run our algorithm with a set of two different deformable objects, a napkin and a small towel. The napkin is 30cm (length) by 30cm (width) and the small towel is 48cm (length) by 28cm (width). While we have performed additional experiments with differently sized planar deformable objects, we omit them in this section due to a limitation of our system. Since the robotic arms are statically mounted and are fairly high from the table, they have a limited workspace. Larger deformable objects would quickly extend past the reachability subspace of our robot and, consequently and legitimately, our algorithm would not find any paths to execute the motion. This drawback is strictly due to our manipulator configuration and could be avoided by a better manipulator placement that would maximize the reachability workspace. The objects are manually placed in front of a robot in such a way that they are within the robot's workspace and are rotated, around their center point, by different angles (from -90 degrees to 90 degrees) to come up with numerous different test cases, resulting in many diverse motions. We choose to run an extensive amount of tests on a virtual representation of our system, simulated in USARSim [5], since it allows to continuously apply the different motions without having a human-in-the-loop, thus greatly facilitating the amount of motion-dependent data that can be acquired. The simulated model faithfully mirrors the real robot. There are only two, negligible, differences between the simulated and real robots. First, the simulated robot is not mounted exactly the same way. More specifically, it is 10 centimeters closer to the table thus giving a larger reachable space. Second, and less importantly, the rotational speed of the joints do not match those of the real robot. It is worthwhile to note that the code implementing the aforementioned algorithm is impervious to the type of robot used (i.e. simulated or real) and that the only difference is the slightly modified robot configuration file. This is consistent with our parallel ongoing research about robot simulators [1]. We invite readers to watch our accompanying video, which shows the same motion plan being executed by the simulated and real robots.

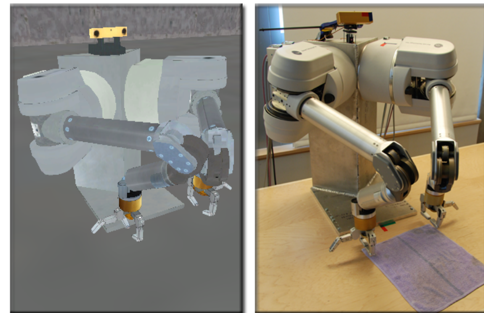

Fig. 3. Juxtaposition of the simulated (left) and real (right) robots at the beginning of the same folding motion. The object is the small towel and is not rotated.

A. Path Generation

In this experiment, we look at the number of collision-free paths that our algorithm is capable of generating. Since the number of paths is highly dependent on the object's configuration and placement relative to the robot, we run experiments using the two objects, the small towel and the napkin, rotating each from -90 degrees to 90 degrees with 5 degrees increments. Consequently, we have two sets of 37 experiments. We use 0.5, 0.5, and 0 for α, β, and γ, respectively. While we have run the experiment for both the simulated and real platform, we only show the results of the simulated data in Figure 4. The real platform's result followed the same shape but yielded, in general, a lower number of collision-free paths, a fact that can be attributed to the manipulators' higher mounting point, reducing their workspaces. The figure shows an almost symmetric pattern between the positive and negative rotations. Even though one might expect the graph to be perfectly symmetric around the 0 degree rotation (e.g. the same series of configurations should be used by the left arm at 10 degrees than the ones used by the right arm at -10 degrees), the arms are not mounted symmetrically from each other (one is rotated by 90

degrees while the other by -90 degrees) and their joint limits are not necessarily symmetric (e.g. joint 4 goes from 180 to -50 degrees). The figure also shows, as expected, a significant difference between the two objects. Generally speaking, the small towel has a greater number of collision-free paths than the napkin, an outcome explicitly explained by their sizes. The napkin being smaller than the small towel forces the manipulators to be positioned closer together when executing the trajectory, resulting in a lot more collisions. Last but not least, the algorithm generates a huge amount of collision-free paths (between 100 and 1000), which allows for either a fine grain selection of the best one or opens a door to make the algorithm faster by reducing the number of chosen paths.

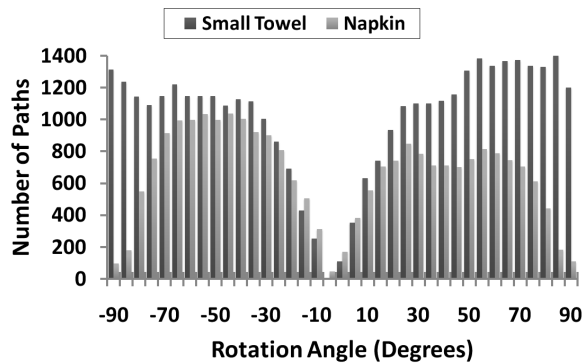

Fig. 4. Number of collision-free paths generated by the algorithm as a function of object rotation angle for the napkin and the small towel.

B. Effect of Time Weight on Motion Execution Time

The proposed algorithm uses weights to dictate how the fold will be executed. In this experiment, we evaluate the effect that the time weight factor, α, has on the overall execution time of the motion plan. Data presented in this section refers to the simulation. We did run, however, similar motions on the real robot (although, a lot less) and have noticed similar patterns to those presented. This should come to no surprise since the time weight is based on the amount of joint rotation. For a given object and rotation, the time weight factor, α, is changed from 0 to 1 with increments of 0.05, each time running the best motion in simulation and recording the total execution time. The collision weight factor, γ, is set to 0 and the boundary singularity weight factor, β is set to $1-\alpha$. Figure 5 shows a few representative examples of the results gathered from this experiment. The graph shows that increasing the time weight factor reduces the overall execution time of the motion by a factor of 18 to 22 percent for the napkin and 30 to 38 percent for the small towel. Folding the napkin takes less time than folding the small towel, a logic observation since the napkin is smaller than the small towel. As a result, the execution times of the small towel can be improved more significantly than those of the napkin.

C. Effect of Collision Weight on Manipulator Distance

In this experiment, we focus our attention on the collision weight factor, γ, to see if it helps force the manipulators keep

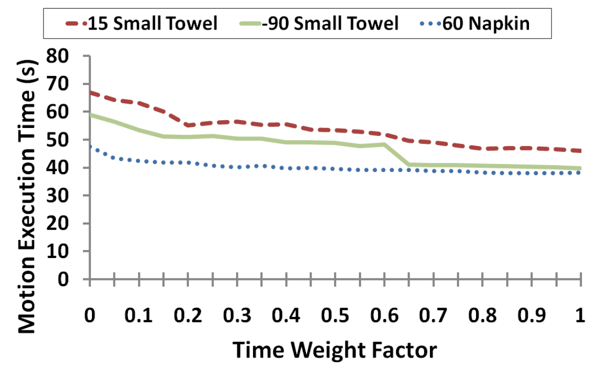

Fig. 5. Motion execution time as a function of the time weight factor, α. Data is shown for the small towel rotated by 15 and -90 degrees and the napkin rotated by 60 degrees.

a safe distance from each other. Similarly to the previous experiment, for a given object and rotation, the collision weight factor, γ, is changed from 0 to 0.95 with increments of 0.5, each time recording the minimum distance between the two manipulators, as given by the collision detector. The time weight factor, α, and the collision weight factor, γ, are both set to $(1-\gamma)/2$. We note that we cannot increase the collision weight factor all the way to 1 since the other two weight factors will be 0, resulting in a cost of 0 for every edge of the graph. Figure 6 shows a few representative examples of the results gathered from this experiment. Counter-intuitively to what one might expect, the minimum distance between the two manipulators does not change significantly. This otherwise peculiar observation can be explained by our starting position that, as noticed earlier, affects the rest of our motion when using the time weight (i.e. rewarding minimum rotations). In other words, our starting position happens to be set in such a way that the time weight produces motions that, indirectly, maximize the arms' distances from each other (e.g. the elbows are forced to face away from each other).

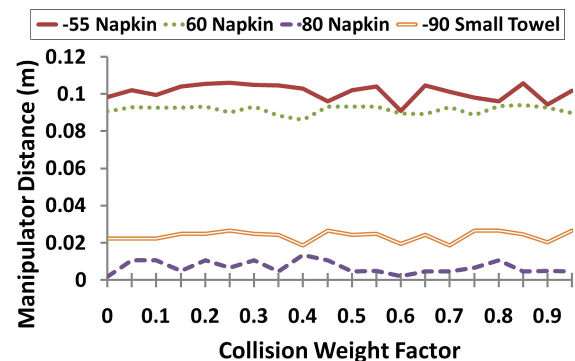

Fig. 6. Minimum distance between the two manipulators as a function of the collision weight factor, γ. Data is shown for the napkin rotated by -55, 60, and -80 degrees and the small towel rotated by -90 degrees.

Readers might wonder why, irrespectively of the object, the motions perpendicular to the robot (e.g. 80 and -90 degrees in Figure 6) result in closer manipulators than for

motions more parallel to the robot (e.g. -55 and 60 degrees in the Figure 6). This difference is explained by the fact that, for perpendicular motions, the elbow of the manipulator closest to the robot has to point away from itself, in the direction of the other manipulator and, as a result, are very close together (see Figure 7). Conversely, more parallel motions allow the manipulator's elbow to stay away from the other manipulator.

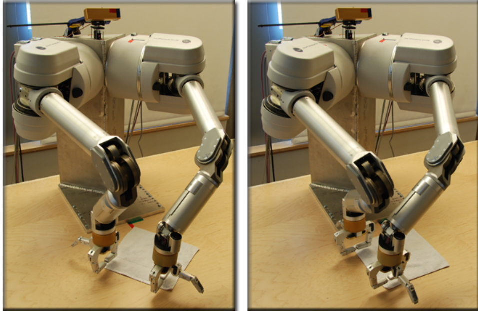

Fig. 7. Pictures showing the difference between perpendicular motions (right) and more parallel motions (left). The perpendicular motion forces the manipulators to be closer to each other.

D. Algorithm Time

We conclude our experimental section with information about our algorithm's running time under two different conditions, the result of which can be found in Table I. The algorithm times were recorded on an Intel Quad Core 2.8GHz desktop computer and include the time spent on allocating space for all the data structures. When $\gamma \neq 0$, the algorithm is much slower since a lot more calls to the collision detector are required in order to find the minimum distance between the two manipulators. However, we have shown in the previous experiment that using the collision weight did not provide any significant improvements. Consequently, we recommend setting γ to 0, unless safety is of outmost importance for the task at hand, to produce a much faster algorithm. The other parts of the algorithm are constant with respect to γ and relatively fast.

Algorithmic Part	$\gamma = 0$	$\gamma \neq 0$
Trajectory Generation	< 1ms	< 1ms
IK Calls	74	74
IK Time	190ms	191ms
Graph Creation	588ms	587ms
Dijkstra Calls	79	79
Dijkstra Time	274ms	275ms
Collision Calls	165	815
Path Selection	42ms	4000ms
Total Time	1.09s	5.05s

TABLE I

TIMING INFORMATION FOR EACH PART OF THE ALGORITHM.

V. Conclusions and Future Work

We have presented a motion planning algorithm capable of generating folding motions for rectangular planar deformable objects. The algorithm's strengths come from its speed, the notion of parameterized fold quality, and the extendibility to different manipulators provided they have at least 6 DOF and perform pinch grasps. We have executed many experiments both in simulation and on the real robotic platform, a subset of which are presented in this paper, corroborating some assumptions while elucidating others. A few different directions can be taken for future work in this area. First, a grasp planner for cloth-like objects needs to be incorporated in order to be able to physically grasp the objects. Second, it would be beneficial to calculate the position of the object that would give the robot the highest chance of generating a motion plan for it. Similarly, and for static robots, we could calculate the position that they should be placed at such that their folding (or other task) capabilities are maximized. We see this paper as the first step towards a fully functional cloth folding robot.

Acknowledgments

We thank Roger Sloan for his help with the collision detector implementation. This work is partially supported by the National Science Foundation under grant BCS-0821766.

References

[1] B. Balaguer, S. Balakirsky, S. Carpin, and A. Visser. Evaluating maps produced by urban search and rescue robots: Lessons learned from robocup. *Autonomous Robots*, 27(4):449–464, 2009.

[2] D. Balkcon. *Robotic Origami Folding*. PhD thesis, Carnegie Mellon University, 2004.

[3] D. Baraff and A. Witkin. Large steps in cloth simulation. In *SIGGRAPH*, pages 43–54, 1998.

[4] M. Bell and D. Balkcom. Grasping non-stretchable cloth polygons. *International Journal of Robotics Research*, 2009.

[5] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper. USARSim: a robot simulator for research and education. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1400–1405, 2007.

[6] L. Chang and N. Pollard. Video survey of pre-grasp interactions in natural hand activities. In *Workshop on Understanding the Human Hand for Advancing Robotic Manipulation at RSS*, 2009.

[7] M. Desbrun, P. Schroder, and A. Barr. Interactive animation of structured deformable objects. In *Graphics Interface*, 1999.

[8] M. Gharbi, J. Cortes, and T. Siméon. Roadmap composition for multi-arm systems path planning. In *International Conference on Intelligent Robots and Systems*, pages 2471–2476, 2009.

[9] S. Gibson and B. Mirtich. A survey of deformable modeling in computer graphics. Technical report, Mitsubishi Electric Research Laboratories, 1997.

[10] S. Gupta, D. Bourne, K. Kim, and S. Krishnan. Automated process planning for robotic sheet metal bending operations. *Journal of Manufacturing Systems*, 17(5):338–360, 1998.

[11] L. Lu and S. Akella. Folding cartons with fixtures: A motion planning approach. *IEEE Transactions on Robotics and Automation*, 16(4):346–356, 2000.

[12] S. Platt and N. Badler. Animating facial expressions. *Computer Graphics*, 15(3):245–252, 1981.

[13] T. Sederberg and S. Parry. Free-form deformation of solid geometric models. In *SIGGRAPH*, pages 151–160, 1986.

[14] G. Song and N. Amato. A motion planning approach to folding: From paper craft to protein folding. *IEEE Transactions on Robotics and Automation*, 20(1):60–71, 2004.

[15] Y. Tsai and H. Huang. Motion planning of a dual-arm mobile robot in the configuration-time space. In *International Conference on Intelligent Robots and Systems*, pages 2458–2463, 2009.

[16] N. Vahrenkamp, D. Berenson, T. Asfour, J. Kuffner, and R Dillmann. Humanoid motion planning for dual-arm manipulation and re-grasping tasks. In *International Conference on Intelligent Robots and Systems*, pages 2464–2470, 2009.

[17] O. Zienkiewicz. *The Finite Element Method Set*. Butterworth-Heinemann, 2005.