Motion Planning for Cooperative Manipulators Folding Flexible Planar Objects

Benjamin Balaguer and Stefano Carpin

***Abstract*— Research on robotic manipulation has mostly avoided the grasping of highly deformable objects, although they account for a significant portion of everyday grasping tasks. In this paper we address the problem of using cooperative manipulators for folding tasks of cloth-like deformable objects, from a motion planning perspective. We demonstrate that complex deformable object models are unnecessary for robotic applications. Consequently, a simple object model is exploited to create 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 demonstrates the method's effectiveness.**

I. Introduction

Still in its early stages, research on manipulation has focused primarily on sensor fusion, the incorporation of a manipulator onto a moving platform, or a single manipulator either grasping simple rigid objects or planning around densely populated spaces using the popular RRTs [10] and PRMs [9]. Additionally, the use of simulation has become very popular, to the detriment of algorithms running on real platforms. In this paper, we take an unprecedented first step in solving what we believe to be some of the most limiting issues: the lack of research involving cooperative manipulators functioning collectively with 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. Indeed, while most objects in our world are inherently deformable, robotics researchers have mostly skipped over the issue of grasping deformable objects or have relaxed the definition of deformable by dealing with near-rigid objects (e.g. a bottle). Evidently, we cannot tackle all of aforementioned tasks and, as such, choose to focus our attention on a simple folding task of a cloth object. A more detailed description of the task is provided in the next Section.

We start by reviewing our problem definition in section II, highlighting any assumptions made. In section III, we describe previous work associated with the deformable models, folding in robotics, and dual manipulator motion planning. Section IV covers our motion planning algorithm, followed, in section V, by experiments. Concluding remarks, extensions, and future work are found in section VI.

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

II. 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 due to their geometrical shape and specific color. 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].

III. Related Work

A detailed review of deformable object models is beyond the scope of this paper and, due to space limitations, we briefly mention the most popular models. Interested readers are pointed to the survey in [8] 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, topological, or material 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, 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. The real robot implementation is inadequate, however, since 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 that the this 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 [7]. 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.

IV. Motion Planning

A. Trajectory Generation in Configuration-Space

Generating a trajectory for each manipulator is tied to the deformable object model. Similarly to [14], [11], we choose a geometrical approach specifically invented for robotic folding, where a fold is represented as a kinematic chain comprised of joints (i.e. creases) connected to rigid links (i.e. faces). Given the length S, width W, object angle A, and corner point $R = Rx, Ry, Rz$ in Cartesian coordinates, we formulate mathematical equations for each trajectory in Cartesian coordinates. We uniformly sample data points from the trajectory by using a variable V, ranging 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 data point density. 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 $Pz = Rz$.

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

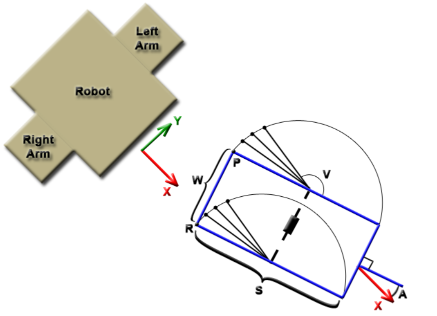


Fig. 1. Geometrical diagram used to derive the trajectories in Cartesian space, showing each mathematical variable.

The data points generated in Cartesian coordinates need to be converted to the robot's C-space. Let $L = \left[L_1 L_2 \cdots L_N\right]$ 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 kth 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

With each trajectory point in C-space, we now focus on building a motion planning graph. Similarly to [7], we generate two graphs, one for each manipulator. Each vertex represents a robot configuration and each edge represents a path from one configuration to another. Each ”level” of the graph corresponds to a distinct Cartesian coordinate, L_i. In other words, each level encompasses many vertices, all representing the same L_i. Each vertex in a level is connected to all the vertices of the next level. A path that follows the trajectory is generated by moving from one level to the next (i.e. moving from one trajectory data point to the next). An initial and final vertex, C_I and C_F, are added as the first and last level of the graph, respectively. A graphical representation of one roadmap is shown in Figure 2.

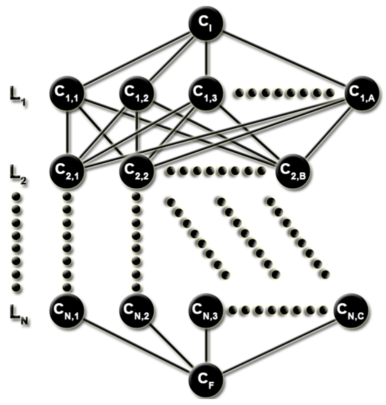


Fig. 2. Graphical representation of a roadmap.

To find the best path within this dense graph, we associate weights to edges. More specifically, we have defined time, boundary singularity, and collision weights as possibilities for definitions of a ”best path”. To minimize the number of calls to the collision detector, the collision weight is implemented in the path selection process (next section). 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 the edge cost is that users can scale the weights based on their definition of ”best path”. As an 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 use the fact that we rotate the robot's joints at a constant velocity. This reduces the problem to minimizing the amount of rotations exerted on the joints. Specifically, given an edge between two configurations, $C_{x,y}$ and $C_{x+1,z}$, we find the joint with the maximum rotational change, which will take the longest to rotate into position. The maximum difference between the positive and negative joint limits (pL and nL) 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 [7], we find the best paths in each roadmap and merge the paths rather than merging the two roadmaps. This procedure limits calls to a collision detector, an important feature of our algorithm, especially given our dense graph. We have two weighted graphs and wish to find the best combination of collision-free paths. We start by finding the best t and u paths of the first and second graph, respectively, by running Dijkstra's shortest path algorithm starting with C_I and C_F as starting and goal configurations. 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 disconnected). We acknowledge that there are different ways of pruning the graph to generate new paths and have found this technique to be successful in finding paths that are different from each other. Simply removing one, or a few, edges would result in paths too similar from each other. Once again, we have a tradeoff between speed and solution density but our pruning method provides a good balance between the two, generating anywhere between 20 and 50 paths per graph. Running the process on each graph results in two sets of paths, the permutation of which gives the total number of solutions. We propose two methods to choose a path from this set. In the first, the set of solutions is ranked by ascending costs and a collision detector determines the first collision-free path. In the second method, we include a new weight, W_c, to account for 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 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 vertex and incorporated into the edge cost, to create a new cost function $E_c = \alpha \times W_t + \beta \times W_b + \gamma \times W_c$, with $\alpha + \beta + \gamma = 1$. With the new costs, the paths are 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.

V. Experimental Results

We performed a series of experiments on our robotic platform, a static torso with two Barrett Arms and Hands, in simulation and on the real system (see Figure 3). For the experiments that we present in this section, we run our algorithm with 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 more 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, 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 paths to execute the motion. This drawback is strictly due to our manipulator configuration and could be avoided by maximizing the reachability workspace when mounting the arms. The objects are placed in front of the robot such that they are within the workspace and are rotated, around their center point, by angles varying from -90 to 90 degrees to come up with numerous different test cases, resulting in many diverse motions. We also chose to run 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. 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. 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.

A. Accuracy

Our first series of experiments attempt to validate how well the robot follows a generated trajectory. This experiment can only be performed in simulation since we need to record the position of the finger tip at any given time. Capturing the finger tip's position of the real robot would require a motion capturing system, plagued with the typical tracking problems due to potential occlusions, which are very likely in our scenario. We execute a series of 49 motions, randomly selecting one of the two objects and rotating it by a random amount, with each motion being comprised of 37 data points. Each time the manipulators reach a data point, the finger tip position is recorded and compared to the desired position. This means that, for this experiment, we are comparing 1813 data points for each arm (i.e. 3626 data points total). As a side note, this exemplifies the potential power of simulations, where running the same series of tests on a real platform would have taken both a longer amount of time and a higher level of human supervision. No outliers were found in this relatively large data set and the root mean square error (RMSE), between the actual and desired position, is 1.6703cm and 1.6385cm for the right and left manipulators, respectively. Even though readers might be tempted to think that the simulation should follow the paths exactly (i.e. the RMSE should be 0 for both arms), we have to keep in mind that imperfections in the simulated model are bound to occur. If the joints are not placed in exactly the same locations as the real robot, discrepancies will occur due to small differences between the DH parameters. These discrepancies account for the aforementioned RMSE.

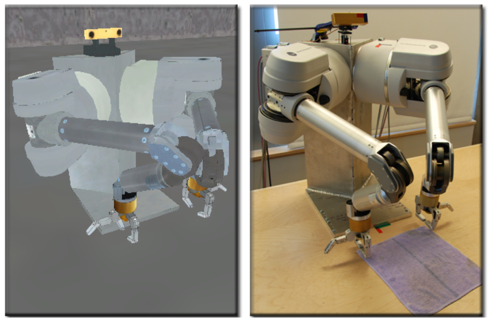


Fig. 3. The simulated (left) and real (right) robots performing the same folding motion. The object is the small towel and is not rotated.

B. 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.

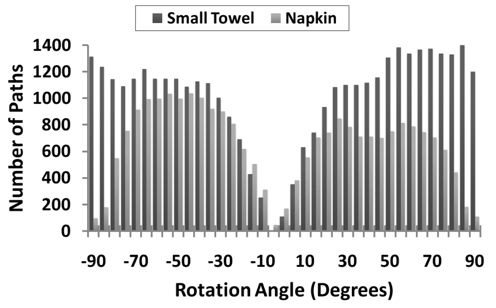


Fig. 4. Number of collision-free paths generated by the algorithm as a function of object rotation angle for the napkin and the small towel.

C. 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. For the same reasons as the accuracy experiment, namely that running many consecutive motions is faster and less human intensive in simulation than on a real robot, the 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.

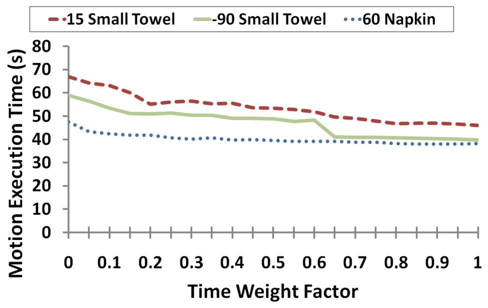


Fig. 5. Motion execution time as a function of α. Data is shown for the small towel at 15 and -90 degrees and the napkin at 60 degrees.

D. 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 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 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 explained 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).

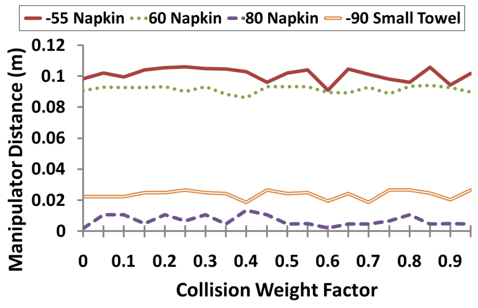


Fig. 6. Minimum manipulator distance as a function of γ, shown for the napkin at -55, 60, and -80 degrees and the small towel at -90 degrees.

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. Figure 7.

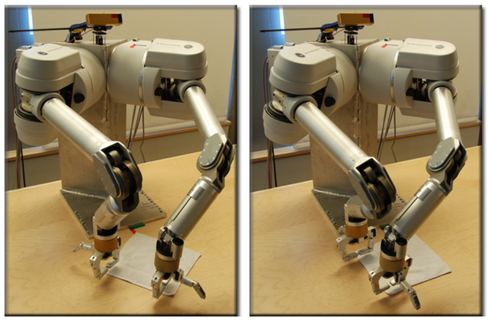
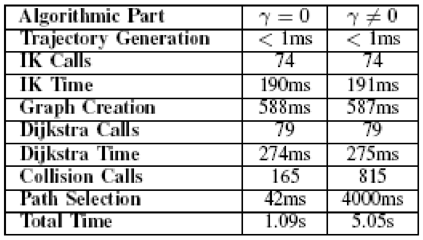


Fig. 7. Pictures showing the difference between perpendicular motions (right) and more parallel motions (left).

E. 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 Figure 8. 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. However, we have shown that using the collision weight did not provide any significant improvements. Consequently, we recommend setting γ to 0, unless safety is of outmost importance. 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

Fig. 8. Timing information for each part of the algorithm.

VI. 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. 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.

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

[9] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[10] S. LaValle and J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In *International Workshop on the Algorithmic Foundations of Robotics*, 2000.

[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.