Folding Arbitrary 3D Shapes with Space-Filling Chain Robots: Folded Configuration Design as 3D Hamiltonian Path through Target Solid

J. Bachrach, V. Zykov, S. Griffith
Makani Power Inc
2175 Monarch Street, Alameda, CA, 94501 USA
jrb@pobox.com, victor@makanipower.com, saul@makanipower.com

Abstract

We propose a method for arbitrary 3D shape formation by folding chains of hinged polyhedra. This paper presents an incremental algorithm for finding a space filling curve, a Hamiltonian path, within a 3D solid. The algorithm is a bottom up approach that starts with minimal local paths connecting small clusters of neighboring constituent polyhedra and incrementally merges neighboring paths along compatible polyhedra edges. We have developed the algorithm for the chains of joined cubes and tetrahedra and show results of chain folding for a number of solids.

1 Introduction

Automatic formation of irregular 3D objects is difficult in practice. In contrast, Biology achieves this routinely by folding sets of chained parts into geometric patterns dictated by mutual components' interactions [2], [10], [9], [6], [7], [1]. This turns out to be a powerful approach that allows for error correction and replication as well as reliable construction of complex physical 3D objects.

In this paper, we describe a method for target configuration design for folding chain robots into space-filling 3D solids. We use a particular chain robot design especially conducive to forming 3D solids through folding: our robots are composed from multiple space-filling polyhedra, such as cubes interconnected with universal joints and right angle tetrahedra interconnected with hinge joints.

Folding chains of hinged polyhedra into arbitrary 3D solids requires that we complete the following two tasks: (1) take a description of a target solid, for example, in the form of a closed triangular surface mesh and produce a final configuration including target positions for monomers and joint angles for the joints so that the chain forms a Hamiltonian path through the shape, and (2) plan for folding the monomers from an unfolded configuration into the final configuration. Specific procedures include rasterization of a target 3D shape, generation of a Hamiltonian path that visits all internal voxels of the target shape, along which the chain robot segments will be lined up in the folded configuration, and, finally, design of a folding trajectory for the chain robot.

This paper presents a set of algorithms and software tools involved in step (1) of rasterization and Hamiltonian path formation. A companion paper [3] describes the tools and algorithms for step (2) of fold trajectory planning.

We are currently focussing on two space packing parts: cubes and right-angled tetrahedra. We are using two universal joints to connect neighboring cube modules and two hinge joints to connect neighboring tetrahedral modules. Figure 1 shows a chain of cubes and a chain of tetrahedra.

2 Related Work

Modular robots are traditionally classified into lattice, chain/tree, and mobile architectures [11]. Lattice robots have traditionally been best known for shape versatility, whereas chain/tree architectures were considered a better fit for manipulation and locomotion tasks [8]. Our approach shows how a chain robot can successsfully be utilized for arbitrary 3D shape formation, thus combining important advantages of the two primary modular robot architectures. Griffith [5] proposed a Hamiltonian path discovery technique based on finding a spanning tree and then finding a path around the spanning tree. Demaine [4] proposed a related idea in constructing hinged polyhedra.

3 Approach

The steps are to rasterize and generate a Hamiltonian path. We consider each step with both the cube and tetrahedral monomers.

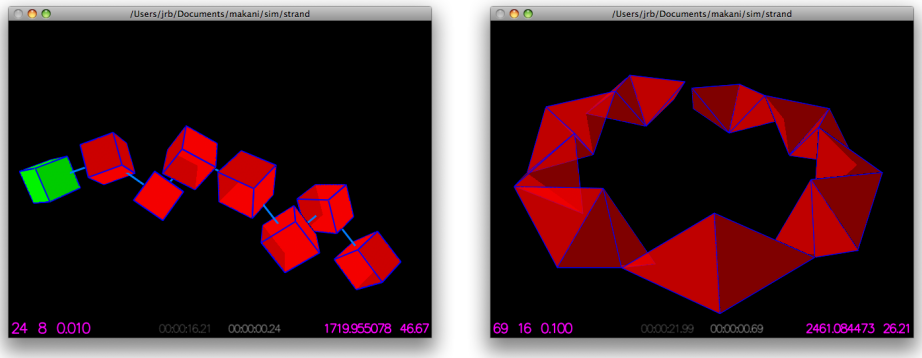

Figure 1. Chain of cubes connected by universal joints on left and chain of tetrahedra connected by hinge joints on right.

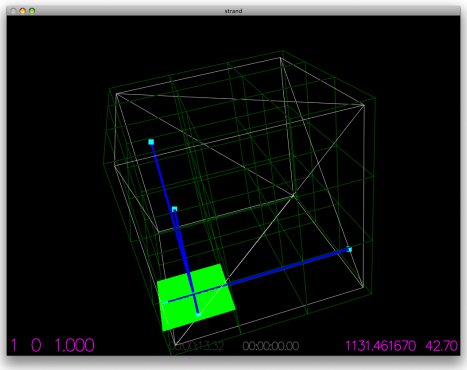

Figure 2. Grid forming cubic voxels.

```
compute_level_set (obj, grid)
  foreach idx in grid
    mn <- inf
    foreach dir in pos_neg_axes
      ints <- find_intersections
                (obj, pos_of(idx), dir)
      mn <- min(mn, min_all(ints))
    grid[idx] = mn
```

Figure 3. Pseudocode for computing level set.

3.1 Rasterization

Space can be broken up into pixels for 2D and voxels for 3D, where pixels/voxels are the cells in a 2D/3D grid respectively as shown in Figure 2. Rasterization is the process of placing a grid over a given shape and determining which voxels are inside and which voxels are outside of the shape.

For little more effort we can determine for each voxel the minimum distance to the surface with interior voxels assigned negative values, exterior voxels assigned positive values, and boundary voxels assigned zero values. This array of distances is called a level set as shown in Figure 4.

The actual technique for calculating the level set values involves ray tracing. Shapes are input in STL format specifying a triangulated surface on the boundary of a given solid object. Any such triangulated surface file format will do. For every cell in the grid six rays are shot in each of the axis aligned directions while calculating the intersections with the triangulated surface. The minimum distance of these six values is recorded along with its sign according to the parity of the total number of intersections in any one direction. It turns out that an odd number of intersections means the point is inside the shape and an even number means the point is outside.

A more efficient way to calculate the level set is to collect all the intersections in one go for each grid value of each row of x,y, and y,z, and x,z. Each cell is then walked through calculating minimum distance in the current direction and add that to the running mimization. This reduces the algorithm from an $O(n^3)$ to $O(n^2)$ algorithm. Further speedups can be achieved by reducing the number of intersection tests by organizing the triangles in an Axis Aligned Bounding Box (aka, AABB) tree.

Rasterizing cubic voxels is straightforward while rasterizing for right-angled tetrahedral monomers is more challenging. We can simplify the problem by making the tetrahedral voxel be a rhombic hexahedron comprised of six tetrahedra as shown in Figure 5. These new voxels are actually very cube like and by using a simple affine transformation we can transform a cubic grid into a hexahedral grid.

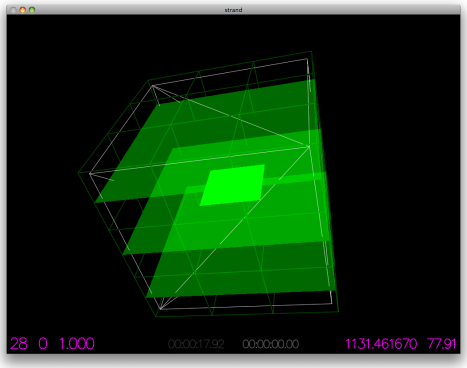

Figure 4. Level values computed over cube

Figure 5. Rhombic hexadron formed out of six tetrahedra.

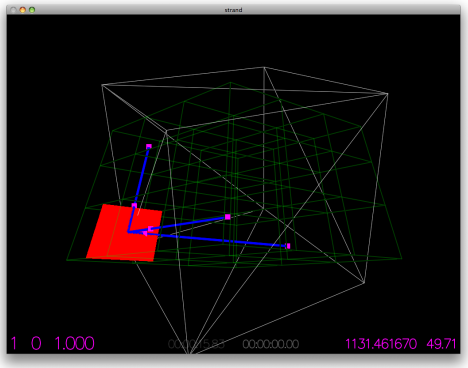

Figure 6. Grid forming rhombic hexahedrals.

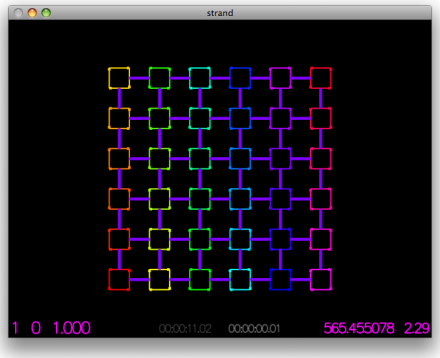

Figure 7. Bottom-up path merging starts with a grid of mini paths.

The affine transformation is:

$$A = mag(1, 1, \frac{1}{2}) \times rot(\frac{\pi}{4}, 0, 0) \times rot(0, atan(\frac{1}{\sqrt{2}}), 0)$$

where mag creates a scaling transformation matrix with x, y, and z components, and rot creates a rotation matrix with x, y, and z components. The rotations are computed and multiplied using quaternions. We can then perform the same level set calculation by stepping through each grid position transformed into hexahedral coordinates as shown in Figure 6. We can determine the grid dimensions by finding the bounding box over the inverse of the affine transformation applied to the set of surface points.

3.2 Bottom-up Path Merging

Our approach to constructing a Hamiltonian path is to incrementally merge increasingly bigger paths. The basic algorithm starts with minimal local paths and incrementally merges neighboring paths. In two dimensions, the algorithm starts with 2x2 squares within the given 2D shape as shown in Figure 7 and then proceeds to merge neighboring squares forming bigger and bigger paths. Eventually the path visits every pixel in the shape. The pseudocode for this algorithm is given in Figure 9. We can actually efficiently maintain the path neighborhood relationship to greatly speed up the algorithm, by having each path maintain its neighbors and only update the neighbors when merged.

Paths can be merged if they share a parallel neighboring line segment. In this case, surgery can be performed on the two paths A and B at the parallel line segments to route the flow between the two paths. In particular, the path would be diverted at the parallel line segments instead going from A to B at the beginning of the segment and then back from B to A at the end of the segment. See Figure 8 for a picture of this operation.

In order to guarantee that we can find 2x2 mini paths, we must first rasterize at half the resolution and then divide the resulting pixels into four subpixels, that is 2x in each dimension. We fondly call these metamodules and having metamodules ensures that there are a complete set of compatible initial mini paths.

This algorithm also works in 3D by starting with mini

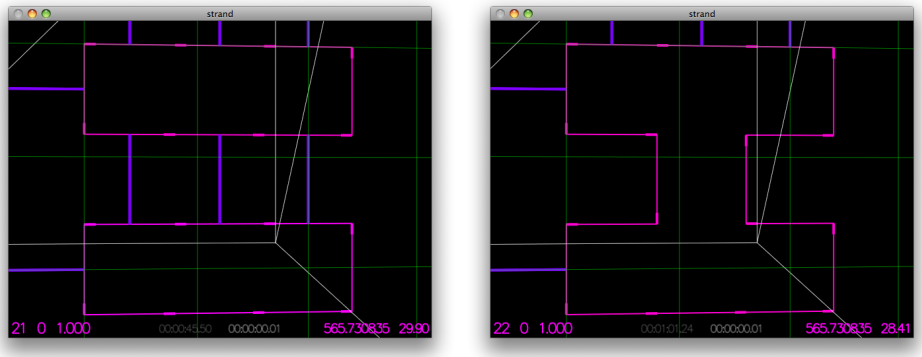

Figure 8. Merging of two paths along a parallel neighboring line segment.

```
build_merge_tour (grid)
  tours <- make_tours(grid)
  while tour_size(tours) > 1
    foreach t1 in tours
      foreach t2 in tours
        if are_tour_nbrs(t1, t2)
          tour_merge(t1, t2)
  return tours[0]
```

Figure 9. Pseudocode for computing Hamiltonian path using undirected incremental merging.

paths over eight neighboring cells. Figure 10 shows one initial mini 3D path. It turns out that if the orientation of the initial mini paths alternate with the parity of $(x + y + z)$ (i.e., 3D checker board) then the algorithm never gets stuck. Figure 10 shows the alternating orientation of mini paths in a 6x6 grid. Notice that all mini paths have at least one compatible edge with their neighbors, unlike the naive implementation with all mini paths having identical orientation. The merging of two paths can never affect the exterior compatibility of remaining paths (including the newly merged path). Figure 11 shows the final path and resulting chain computed from running that algorithm on a 6x6 grid.

Hamiltonian path construction works on the tetrahedral chain by first constructing mini paths along the center of the six tetrahedral modules in each rhombic hexahedron used in the initial rasterization process. Each rhombic hexahedron can be viewed as a squashed cube and as such has six rhombic hexahedron neighbors, where each face is comprised of two tetrahedral faces and one hinge joint as shown in Figure 5. Merging occurs between hexahedral faces and involves an analogous surgery to redirect the tour between the neighboring rhombic hexahedral metamodules.

4 Results

In this section we show a number of examples of the incremental Hamiltonian path creation algorithm. First we ran the algorithm on a series of shapes with the cubic geometry. Figures 13 and 14 show the results of running the algorithm on a wrench and gorilla respectively.

Finally, we ran the algorithm on the tetrahedral geometry. Figure 15 shows the algorithm run on a wrench. The accompanying video shows the algorithm run with the cubic geometry on a square, cube, and wrench and the tetrahedral geometry on a cube.

5 Conclusion

In this paper, we presented an algorithm for incrementally finding a Hamiltonian path through a solid. This path is used in order to discover a target position for a robotic chain tesselating a solid. In [3] we show how we can plan a folding trajectory to fold an unfolded chain into the desired shape. Future work involves directing the merging to form paths more conducive to folding. We will also extend the work to finding space filling curves on surfaces. The cost of the algorithm is that we must have 8x more parts for the cube geometry and 6x more parts for the tetrahedral geometry to allow for the creation of the initial metamodules and mini paths. If we could find a complete set of mini paths in the original resolution, then we could create complete Hamiltonian paths with this algorithm with far fewer parts. We will explore this possibility in the future.

6 Acknowledgements

We would like to thank DARPA for their funding under the Programmable Matter program. We would like to thank Erik Demaine, Neil Gershenfeld, Kenny Cheung, Jim

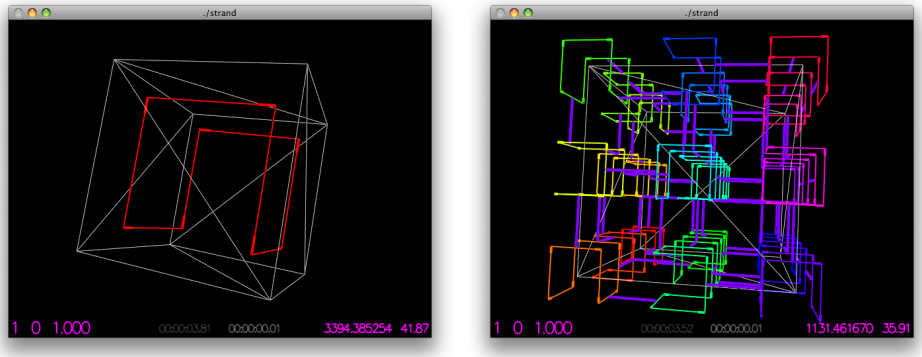

Figure 10. Initial mini path and initial orientation of mini paths.

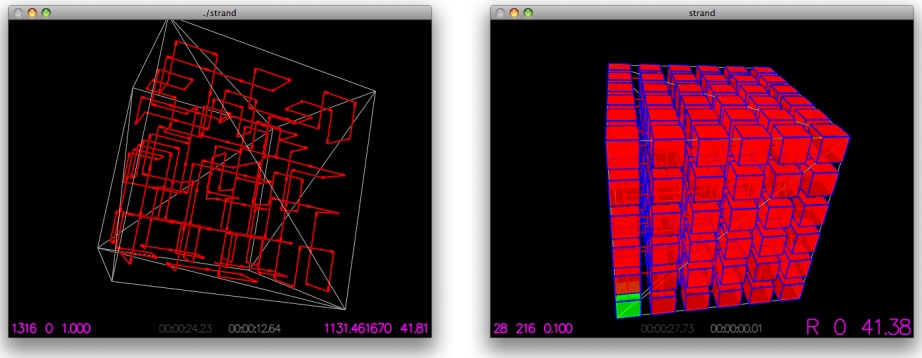

Figure 11. Final Hamiltonian path and resulting 216 cube chain on a 6x6 grid.

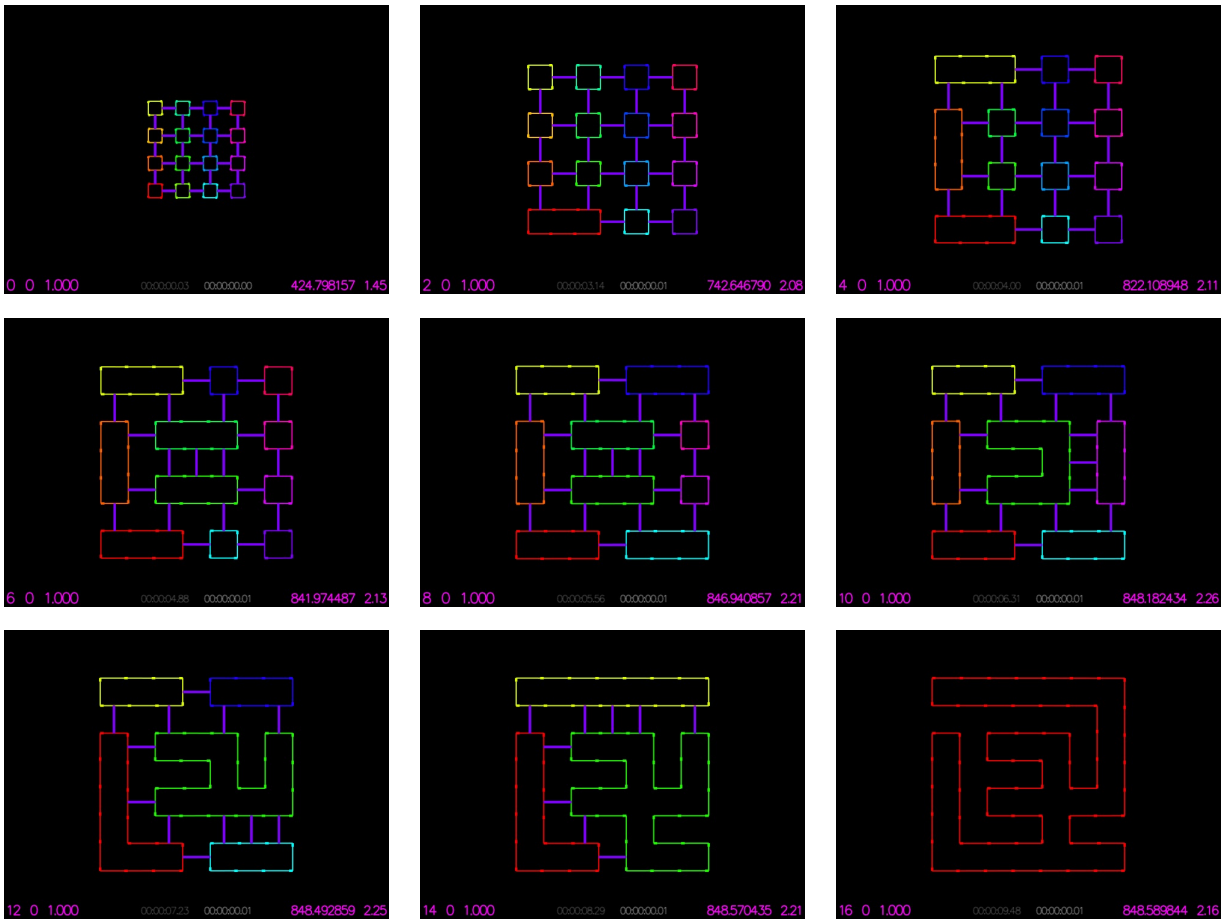

Figure 12. Incremental bottom up path merging on a 8x8 grid filling a square. Each path has a unique color and purple lines between path depict the neighborhood relationship. Some of the merging steps were removed for brevity.

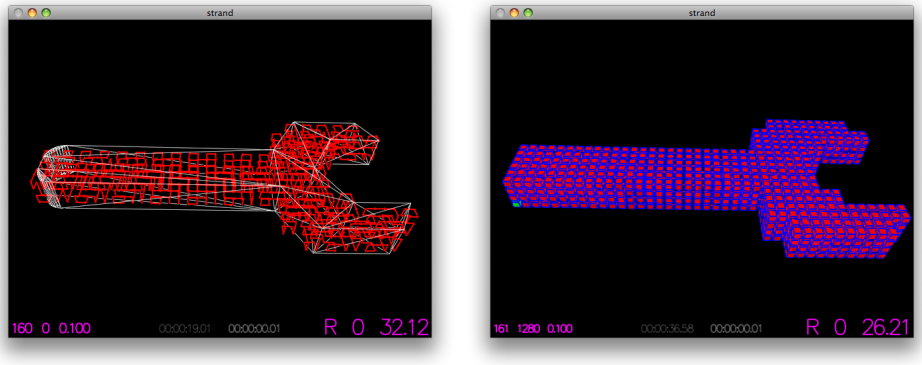

Figure 13. Final Hamiltonian path and resulting 1280 cube chain using cubic geometry on wrench solid.

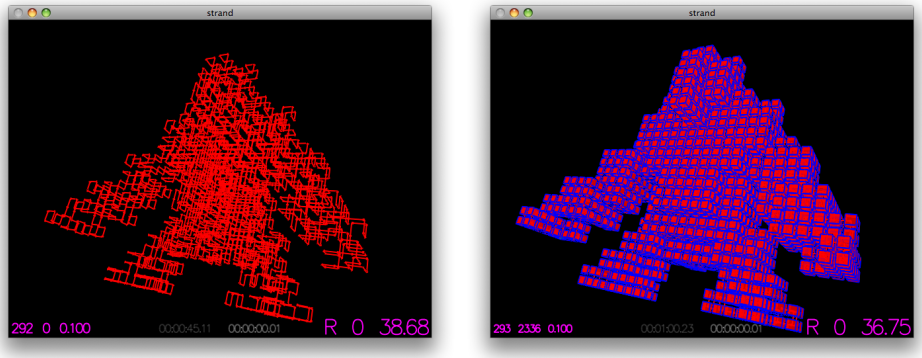

Figure 14. Final Hamiltonian path and resulting 2336 cube chain using cubic geometry on gorilla solid.

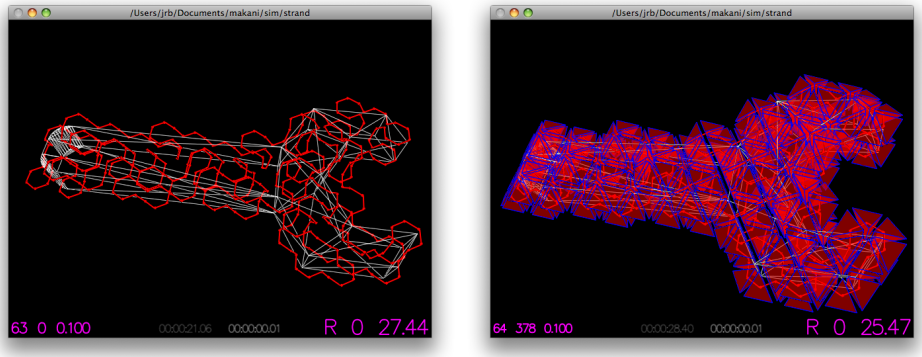

Figure 15. Final Hamiltonian path and resulting 62 tetrahedral chain using tetrahedral geometry on wrench solid.

McBride, Ara Knaian, and the rest of the MIT Millibiology team for their helpful suggestions and guidance.

References

[1] P. Alexander, Y. He, Y. Chen, J. Orban, and P. Bryan. Characterization of protein-folding pathways by reduced-space modeling. In *Proceedings of National Academy of Science USA*, volume 104, pages 11963–8, 2007.

[2] C. Anfinsen. The formation and stabilization of pretein structure. *Journal Biochem*, 128:737–49, 1972.

[3] J. Bachrach, V. Zykov, and S. Griffith. Fold trajectory planning using reverse explosion servoing. Under submission, 2009.

[4] E. Demaine, M. Demaine, J. Lindy, and D. Souvaine. Hinged dissection of polypolyhedra. In *Proceedings of the 9th Workshop on Algorithms and Data Structures*, Lecture Notes in Computer Science. Springer Verlag, August 2005.

[5] S. Griffith. *Growing Machines*. PhD thesis, MIT, 2004.

[6] S. Kmiecik and A. Kolinski. Characterization of proteing-folding pathways by reduced-space modeling. In *Proceedings of National Academy of Science USA*, volume 104, pages 12330–35, 2007.

[7] S. Lee and F. Tsai. Molecular chaperones in protein quality control. *Journal Biochem Molecular Biology*, 38:259–65, 2005.

[8] S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kakaji. M-tran: Self-reconfigurable modular robotic system. *IEEE/ASME Trans. Mech.*, 7(4):431–441, 2002.

[9] C. Pace, B. Shirley, M. McNutt, and K. Gajiwala. Forces contributing to the conformational stability of proteins. *Faseb Journal*, 10:75–83, 1996.

[10] G. Rose, P. Fleming, J. Banavar, and A. Maritan. A backbone-based theory of protein folding. In *Proceedings of National Academy of Science USA*, volume 103, pages 16623–33, 2006.

[11] M. Yim, W. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. Chirikjian. Modular self-reconfigurable robot systems: Challenges and opportunities for the future. *IEEE Robots and Automation Magazine*, pages 43–52, March 2007.