November 1991

Automatic assembly planning and control via potential functions

Louis L. Whitcomb
Yale University

Daniel E. Koditschek
University of Pennsylvania, kod@seas.upenn.edu

NOTE: At the time of publication, author Daniel Koditschek was affiliated with Yale University. Currently, he is a faculty member in the Department of Electrical and Systems Engineering at the University of Pennsylvania.

Automatic assembly planning and control via potential functions

Abstract
An approach to the problem of automated assembly planning and control using artificial potential functions is described. A simple class of tasks, 2D sphere assemblies, is examined. A constructive theory for the planning and control of this class of tasks is presented. Computer simulations demonstrate that the approach may provide surprisingly good performance.

NOTE: At the time of publication, author Daniel Koditschek was affiliated with Yale University. Currently, he is a faculty member in the Department of Electrical and Systems Engineering at the University of Pennsylvania.

Automatic assembly planning and control via potential functions

Louis L. Whitcomb * Daniel E. Koditschek †

*Yale University,

†University of Pennsylvania, kod@seas.upenn.edu

IEEE/RSJ International Workshop on Intelligent
Robots and Systems IROS '91, Nov. 3-5, 1991,
Osaka, Japan. IEEE Cat. No. 91TH0375-6

Automatic Assembly Planning and Control via Potential Functions

Louis L. Whitcomb and Daniel E. Koditschek *
Center for Systems Science
Yale University, Department of Electrical Engineering

Abstract

A new approach to the problem of automated assembly planning and control using artificial potential functions is described. A simple class of tasks, 2-D sphere assemblies, is examined. A constructive theory for the planning and control of this class of tasks is presented. Computer simulations demonstrate the new approach may provide surprisingly good performance.

1 Introduction

The automated assembly problem — planning and controlling the movement of a collection of rigid body parts from arbitrary dis-assembled initial configurations to an assembled final configuration — arises in a multitude of commonplace tasks such as robot assembly of manufactured parts, robot palletizing, and automated warehousing. The task specification consists of a description of the parts (shape, size, dynamical model) accompanied by a parametric representation of the final (assembled) part configurations. We will address three versions of this problem:

1. The *assembly planning* problem is the construction of a collision-free curve in the part configuration space which joins any initial configuration and final point.
2. The *sequential assembly planning* problem is the construction of a configuration space curve which achieves the same objectives while only requiring movement of one object at a time.
3. The *assembly control* problem is the construction of a force control law (for a specified robot manipulator) which effects a collision free part transfer from any such dis-assembled configuration to the desired assembled state.

A recently developed class of artificial potential functions that we have termed *Navigation Functions*, [5], appears to provide, in some instances, a simultaneous solution to both the planning and control problem of automated assembly. If this approach can be generalized to a broad class of assembly tasks, it would possess three desirable properties. First, the problem specification is simply the natural geometric and dynamic properties of the parts rather than high level interpretations of this data such as, for example, an assembly graph[1][1]. Second, it appears that Navigation Function based controllers naturally extend to address the *sequential assembly* problem where a single manipulator (of relatively few degrees of freedom) can only grasp a single part at a time, and thus must achieve a completed assembly via a sequence of individual part manipulations. Finally, Navigation Function based controllers preserve collision-free behavior and would provide the desirable stability and robustness properties characteristic of feedback in contrast to their open-loop counterparts.

This paper reports on a construction and numerical simulations of Navigation Function based solutions to both the *assembly planning* and *sequential assembly planning* problems for a simple environment. The important extension to the *assembly control* problem [2] for this environment is the subject of ongoing work. We identify several outstanding technical issues which must be addressed to put this approach on a provably correct theoretical foundation.

2 A Simple Class of Problems: 2-D Sphere Assemblies

This section describes a simple class of assembly tasks arising from a planar workspace inhabited by a collection of movable rigid disks. In the sequel we will present an algorithm for planning the collision-free movement of the disks from any initial configuration to a desired "assembled" configuration.

Each of n balls, which are free to move, is uniquely specified by its position $b_i \in \mathbb{R}^2$, radius $\rho_i \in \mathbb{R}^+$, and the composite vector of n balls is $b \in \mathbb{R}^{2n}$. Label the *desired* ball positions $d_i \in \mathbb{R}^2$ and $d \in \mathbb{R}^{2n}$ respectively. We necessarily assume both the initial and desired states are "legal" — no balls touch and all reside within the boundary. Figure 1 shows a typical 7 ball assembly sequence from (random) initial configuration to "assembled" final configuration. Figure 2 shows a 10 ball assembly.

It must be emphasized that the 2-D sphere assembly problems (and its solutions) are a vast oversimplification of real-world assembly tasks. Our hope is that techniques employed herein might be extended to richer classes of problems of real practical usefulness. This paper constitute a prelude and the first first modest step in a program of research on automated assembly and control via Navigation Functions.

*This work was supported in part by the National Science Foundation under a Presidential Young Investigator Award held by the second author.

[1]There is no reason to hope, however, that any potential function approach will be computationally more efficient than the alternative techniques.

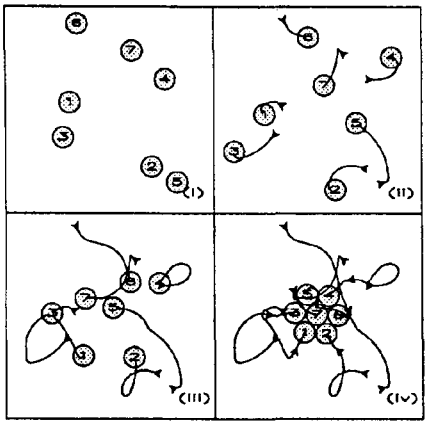

Figure 1: A Seven Ball Assembly Sequence: Balls indicate current configuration b, Lines indicate the ball paths. Frame (i) shows random initial configuration, (ii) and (iii) are intermediate configurations, and (iv) shows final (desired) configuration.

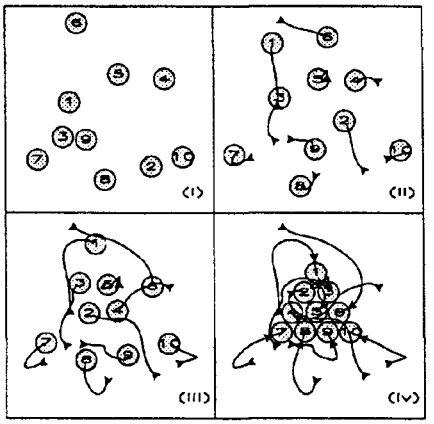

Figure 2: A Ten Ball Assembly Sequence: Balls indicate current configuration b, Lines indicate the ball paths. Frame (i) shows random initial configuration, (ii) and (iii) are intermediate configurations, and (iv) shows final (desired) configuration.

3 A Navigation Function for 2-D Sphere Assemblies

This section describes the construction of what we believe (but have not yet proven) to be a Navigation Function, a term we shall define more precisely, for the class of simple assembly tasks defined in Section2.

3.1 What is a Navigation Function?

We desire a smooth function $\phi : \mathbb{R}^{2n} \rightarrow [0,1]$ from the configuration space of the assembly to the unit interval which (i) attains a uniform maximum value on the configuration space boundary (those configuration space points which correspond to workspace collisions) and (ii) has a ***unique*** global minimum at the desired point in configuration space, all other critical points being isolated saddles. With some additional technical conditions, this describes the class of Navigation Functions [5]. Given a function $\phi(b)$ possessing these properties, the state of

$$\dot{b} = -\nabla\phi(b) \tag{1}$$

will asymptotically approach d, without collision, from all initial conditions ***except*** those within the basin of attraction of the (topologically inevitable) saddle points — a set of measure zero.

3.2 Constructing a Navigation Function

This section describes the construction of a family of functions clearly possessing the first property, but not necessarily possessing the second. Koditschek [2] has recently proven that property (ii) holds for a one-dimensional version of this problem. As the simulations in Section 5 will suggest, this construction ***appears*** to possess the property for the two-dimensional case. We surmise that the property holds in general.

First construct the cost function

$$\alpha(b) = \frac{\gamma(b)^k}{\beta(b)} \tag{2}$$

where $\gamma(b)$ is a smooth function which is positive everywhere except at the point $b = d$ (where it has a value of zero), k is a design parameter, and $\beta(b)$ is a smooth function which is positive everywhere except at those configurations where a ball makes contact with another body (where it has value zero).

The function $\gamma(b)$ is given by

$$\gamma(b) = \sum_{i=o}^{i=n} \gamma_i(b); \qquad \gamma_i(b) = \frac{1}{2} \| b_i - d_i \|^2 \tag{3}$$

and $\beta(b)$ is given by

$$\beta(b) = \prod_{\substack{i=1 \\ j=i}}^{\substack{i=n \\ j=n, \\ i \neq j}} \beta_{b_i,b_j}(b) \tag{4}$$

where the $\frac{n}{2}(n-1)$ functions

$$\beta_{b_i,b_j}(b) = \frac{1}{2}(\| b_i - b_j \|^2 - (\rho_i + \rho_j)^2) \tag{5}$$

are always positive when the balls are not touching, and zero when ball i and j touch.

The function $\alpha(b)$ is zero only at $b = d$, positive elsewhere, and becomes infinite when contact occurs between any two balls. In contrast, a navigation function takes values on the unit interval $[0,1] \in \mathbb{R}$. Thus we limit the magnitude of $\alpha(b)$ (and hence its gradient) by taking the composition with a "squashing" function

$$\phi(b) = \sigma(b) \circ \alpha(b); \qquad \sigma(q) = (\frac{q}{1+q})^{\frac{1}{k}}. \tag{6}$$

The squashing function σ again contains the design parameter k which will be adjusted in the examples to eliminate local minima.

Taking the gradient of ϕ we have[2]

$$\nabla_b \phi = \nabla_b[\sigma \circ \alpha(b)] = \nabla_b[\gamma(\beta + \gamma^k)^{\frac{-1}{k}}] \quad (7)$$

$$= (\frac{1}{k}(\beta + \gamma^k)^{-\frac{1+k}{k}})[k\beta D_b[\gamma]^T - \gamma D_b[\beta]^T] \quad (8)$$

Equation (8) consists of two components. The first, $k\beta D_b[\gamma]$, is "attractive" and points in the direction of the desired destination point in configuration space. The second component, $\gamma D_b[\beta]$, is "repulsive" and points "away" from collisions.

3.3 Normalized Gradient Dynamics

In practice the magnitude of $\nabla\phi$ is often small — values smaller than 10^{-20} are commonly observed even in simple cases. These small gradient norms present two difficulties. First, since the "speed" of the gradient system (1) is precisely $\| \nabla\phi \|$, the system may take an extraordinary amount of time to converge to the desired point d. Second, numerical integration of (1), which requires the addition of quantities which scale with $\| \nabla\phi \|$ to quantities of order $\| b \|$, is limited by machine numerical precision — and can fail[3].

To remedy this problems, we normalize the gradient vector field by taking its product with the function $\lambda = (\epsilon + \| \nabla\phi \|)^{-1}$ where ϵ is empirically selected. The new dynamical system $\dot{b} = -\lambda \cdot \nabla\phi(b)$ is free from the performance problems of (1) described above yet can be shown to inherit its correctness properties.

3.4 Feedback Control Mechanical Systems

While the solutions to the ordinary differential equation (1) represent collision-free paths in configuration space to the desired "assembly", practical applications require a control law relating object states to actuator commands (forces and torques) which will drive the objects to the desired configuration. It is shown in [4] that we can "lift" the gradient dynamical system (1) to provide a feedback controller for the corresponding lagrangian systems which preserves the desirable properties of collision avoidance and stable "assembly". Recent work [3] suggests that it is possible to similarly lift the normalized gradient system.

4 Two Assembly Planning Algorithms

In this section we propose solutions to both the *assembly planning* and *sequential assembly planning* problem for 2-D sphere assemblies. Each algorithm employs (i) the coordinates of a desired "assembled" 2-D ball configuration and (ii) the coordinates of an (essentially) arbitrary initial configuration to construct a navigation function based planning algorithm. The algorithms, each utilizing a set of $\mathbb{R}^{2n}$ dimensional ordinary differential equations generate *a collision free configuration-space path to the "assembled" ball configuration.*

Section 4.1 defines a dynamical system whose solution curves solve the 2-D *assembly planning* problem. Section 4.2 extends and modifies this approach to the more practically useful 2-D *sequential assembly planning* problem.

4.1 The Gradient Assembly Algorithm

We conjecture that $\phi(b)$ (6) is a Navigation Function. If so, then trajectories of the gradient dynamical system

$$\dot{b} = -\nabla\phi(b) \quad (9)$$

define collision free paths to the desired "assembled" configuration-space coordinate d. Thus we propose the evaluation of trajectories of the ordinary differential equation (9) as a solution to the *assembly planning* problem. In the sequel we shall refer to the system (9) as the *gradient assembly algorithm.*

Note that a solution to (9), denoted $b(t)$, is a map from $\mathbb{R}^1$ to $\mathbb{R}^{2n}$. The curve $b(t)$ *simultaneously* specifies the *time-varying* position of every ball, thus describing a choreographed dance in which all balls move simultaneously. They appear to cooperatively jostle their way (without touching!) into the specified "assembled" configuration. Fabricating an actual robot workcell using this algorithm would require all work pieces (balls) to be somehow independently actuated. We will relax this requirement in the following section.

As general closed-form solution to (9) does not exist, in Section 5 we will use numerical integration to carefully examine properties of its solution curves.

4.2 The Sequential Assembly Algorithm

In practice, it is not possible for a single robot arm to *simultaneously* and *independently* manipulate, say, a dozen workpieces. A robot manipulator is typically capable of grasping and manipulating one object at a time. In this section we present an assembly algorithm which generates assembly trajectories for which only *one* part moves at a time. We shall refer to this as the *sequential assembly algorithm* in the sequel. The algorithm repeatedly specifies *which* part the robot should grasp, *where* to move it, and *when* to release it. A sequence of such actions will transfer the ball to the final configuration. Using this strategy we might (i) guarantee successful completion of the assembly and (ii) avoid collisions while (iii) requiring the robot to manipulate only one part at a time.

We now propose a two-level hierarchical algorithm to solve the sequential planning algorithm in which the higher level will sequentially join solutions amongst n different low level algorithms. The resulting curves form a continuous collision-free path in configuration space in which only one ball moves at a time.

4.2.1 Sequential Algorithm: Low Level Part

Define n *low level* differential equations to be simply the projection of the $2n$ dimensional gradient (8) onto the i^{th} sub-

[2] In the sequel we will omit explicit reference to the independent variable b where it is clear from context, and use employ the notation $D_b[f(b)]$ to represent the generalized row-vector (or matrix valued) Jacobian of a scalar (or vector) valued function $f(b)$ with respect to the independent variable b.

[3] For example, recall that in the standard double length IEEE floating point representation, for x smaller than about 10^{-16}, $1.0 + x = 1.0$

space $\mathbb{R}^2$ associated with the i^{th} ball

$$\dot{b} = f_i(b); \quad i \in \{1, ..., n\} \tag{10}$$
$$f_i = S_i \nabla_b \phi \tag{11}$$

where

$$S_i = \begin{bmatrix} 0_{2x2} & \cdots & 0 \\ & \ddots & \\ \vdots & \underbrace{I_{2x2}}_{\substack{i^{th} \\ position}} & \vdots \\ & & \ddots \\ 0 & \cdots & 0_{2x2} \end{bmatrix}. \tag{12}$$

It is important to recognize that solutions to the i^{th} differential equation (10) are it not simply projections of the solution of (9) onto the i^{th} ball's subspace $\mathbb{R}^2_i$. Solutions to equation (10) specify that the i^{th} ball move *individually* along its component of the overall gradient (9) until "blocked" by the other $j \in \{1..n\}, j \neq i$ immobile balls.

4.2.2 Sequential Algorithm: High Level Part

It is the job of the *high level* algorithm to sequentially choose from among the i low level differential equations, evaluate the solution until it "blocks", and then select a different one. With some abuse of notation[4], we can write the high level controller as the discrete dynamical system

$$x_{n+1} = T(x_n) \tag{13}$$

where the $T : \mathbb{R}^{2n} \mapsto \mathbb{R}^{2n}$ is the transition map from one "blocked" ball configuration to the next. T is given by

$$T(x_n) = \Omega_{i_n}(x_n); \; i_n \equiv C(x_n) \tag{14}$$
$$\Omega_i(x) = \lim_{t \to \infty} \{y(t) : \; \dot{y} = f_i(y); \; y(0) = x\} \tag{15}$$

where $\Omega_i(x)$ is function returning the limit point of the i^{th} equation (10) with initial condition x. Finally, the selection function $C(x)$ returns the index of that $f_i(x)$ (11) with the greatest magnitude.

$$C(x) = \{j : \; \| f_j(x) \| = \max_{1 \leq i \leq n} \| f_i(x) \|\}. \tag{16}$$

The resulting ball trajectories, comprising a sequential concatenation of solutions to selected differential equations, are continuous, piecewise smooth, and require only one ball to be moved at a time. Of course the actual process of "switching" workpieces requires that the robot release one object from its gripper and acquire a different object — often a nontrivial process. In consequence, in the simulations we will examine not only the sequential algorithm's pathlength ratio, but the number of switches required to complete an assembly as well.

[4] Here we use $x \in \mathbb{R}^{2n}$ to represent the configuration space state of the high level controller at the i'th sequential step to distinguish it from $b_i(t)$, the x-y position of the i^{th} ball at time t.

5 Simulations

In this section we present the results of computer simulation study of solutions to the assembly planning problem for the simple problem class of 2-D sphere assemblies outlined in Section 2. Section 5.2 examines the performance of the gradient planning algorithm introduced in Section 4.1. Section 5.3 examines the performance of the sequential planning algorithm introduced in Section 4.2.

We have constructed a computer simulation of the above systems consisting of a numerical ODE integration package [6] coupled to an interactive graphical interface which provides animated rendering of the complete dynamical system. The difficulty, and hence performance, of an assembly task varies with initial and final conditions. We have attempted to achieve a better than anecdotal understanding of performance in the following simulations by conducting numerous trials with random initial conditions. We are thus able to report the mean and standard deviations of performance measures as "typical" for a variety of initial conditions.

5.1 A Performance Measure

How can assembly performance be measured? The obvious choice, comparison to "optimal" performance, necessitates the selection of an appropriate performance metric. Candidate metrics might include pathlength, time, energy, computational effort, and the like.

For this initial investigation we have employed perhaps the simplest and most easily measured performance metric — assembly pathlength. Assembly pathlength is the distance (in $\mathbb{R}^{2n}$) traveled by the spheres from initial configuration to desired final "assembled" configurations.

Since pathlength necessarily varies with initial condition, we wish to normalize results with respect to the minimum attainable pathlength. Minimum attainable pathlength, however, is difficult to compute in practice. We have instead normalized the results with respect to the straight-line euclidean distance from the initial to final point, and shall term this measure the "path-length ratio" in the figures. The straight-line path (which might be physically unattainable because of collisions) is, of course, less than or equal to the minimum length collision-free path. The smaller the path-length ratio, the better the algorithm's performance. Thus, for example, a path length ratio of 2.0 occurs in the case where the assembled bodies travel exactly twice as far from start to finish as would have been required by a straight-line path in configuration space.

5.2 Performance of the Gradient Assembly Algorithm

In this section we examine two performance aspects of the gradient assembly system. Section 5.2.1 examines the effect on path-length of the system parameter k. Section 5.2.2 then examines the effect of assembly "difficulty" on path-length performance.

5.2.1 The Effect of the Parameter K on Pathlength.

The parameter k, which originates as an exponent in (2) and (6), enters as a scaling coefficient of the "attractive",

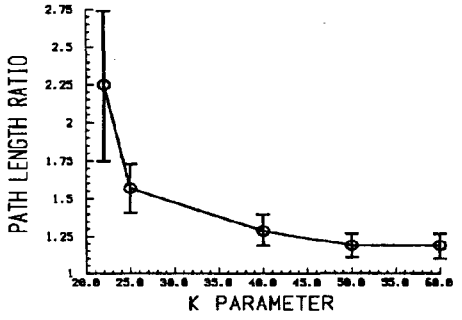

Figure 3: Path Length Ratio vs k for the 7 ball assembly of Figure 1. Each data point represents the Mean and Standard Deviation of 25 runs with random initial configuration.

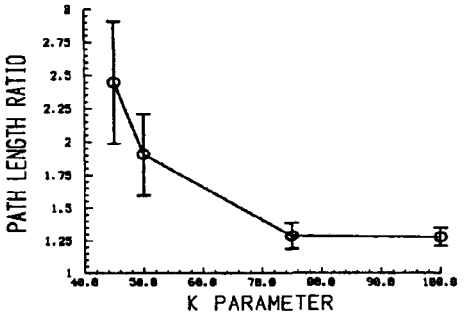

Figure 4: Path Length Ratio vs k for the 10 ball assembly of Figure 2. Each data point represents the Mean and Standard Deviation of 25 runs with random initial configuration.

$k\beta D_b[\gamma]$, versus "repulsive", $\gamma D_b[\beta]$, components of (8). Earlier work on a point moving amongst sphere obstacles concludes that ϕ, (6), becomes a navigation function when k is sufficiently large. We have observed in simulation (and are in the process of proving rigorously) that values of k above a context-defined threshold ensure solutions to (9) converge to the desired point d. Conversely, we have also observed solutions to (9) do not converge to d for values of k below this threshold. It is important to understand, in the former case, how variations in k effect path-length performance.

Figure 3 shows the path-length ratio mean and standard deviations (for 25 random initial conditions) at five different values of k for the seven ball assembly of Figure 1. Higher values of k clearly yield smaller mean path-length ratios and, from the proportionally smaller standard deviations we may conclude this relationship is "typical". In this example we see mean path-length ratios as low as the 1.25 (for the highest k value), indicating that solutions to (9) have average length only 25% greater than the straight-line path. Indeed, we have observed similar performance figures for a great variety of assemblies.

Figure 4 shows the path length ratios for the more complicated ten ball assembly of Figure 2. Here again we observe the path length ratios to be under 1.25 for high k values. This is remarkable considering solutions to (9) are *guaranteed* to be collision-free, while the (only marginally shorter) straight-line path may result in collisions, making it physically unrealizable.

5.2.2 The Effect of Assembly "Difficulty" on Pathlength.

We naturally expected path-length performance to vary with the "difficulty" of the assembly task — that the closely packed desired assembly of Figure 5(d) would be more difficult to attain than a loosely packed assembly of figure 5(a). The function $\beta(d)$, (4), is the product of all pairwise distances between objects. Figure 5 shows that $\beta(d)$ varies in a manner that corresponds to our intuitive notion of assembly difficulty — The tightly packed assembly of Figure 5(d) has $\beta(d)$ value of 10^{-44} while an "easier" loosely packed assembly of 5(a) has $\beta(d)$ value 10^{-2}. More closely packed (harder) assemblies have smaller $\beta(d)$.

Figure 6 shows the resulting plot of path-length ratio

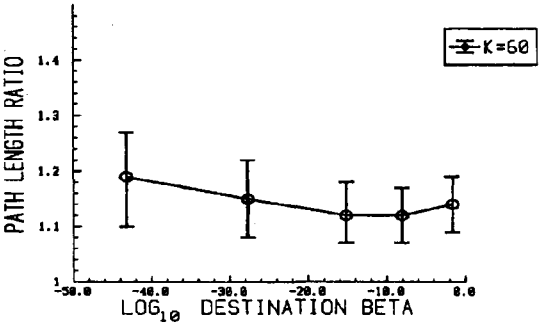

Figure 6: Path Length vs $\beta(d)$ for each of the five 7-ball assemblies of Figure 5. Each data point represents the Mean and Standard Deviation of 25 runs with random initial configuration.

(mean and standard deviation for 25 runs) for five different values of destination beta. Surprisingly, the path-length ratio appears to be nearly independent of the assembly $\beta(d)$ value. "Harder" assemblies (having smaller $\beta(d)$) result in path-length ratios only marginally greater than for "easy" (larger $\beta(d)$) assemblies.

Figure 7 shows the result of the same experiment of Figure 6 repeated for five different values of k. This rather complicated figure demonstrates two important points. First, the the insensitivity of path-length ratio to assembly "difficulty" holds for a wide range of k values. Second, the remarkably good pathlength performance reported in the previous section is observed to hold for both easy (large $\beta(d)$) and hard (small $\beta(d)$) assemblies.

Of course, path length ratio also varies with the "difficulty" of the *initial* configuration of the assembly parts. We are particularly interested in examining the performance of these algorithms in situations where several pieces need to be moved aside before other pieces can be moved into their designated position. Thus, we also ran a series of simulations whose destinations were tightly packed (small $\beta(d)$), and whose 25 initial ball positions were random permutations of the desired final positions. This resulted in path-length ratios nearly double those observed for random initial configurations but with trends otherwise similar to the less

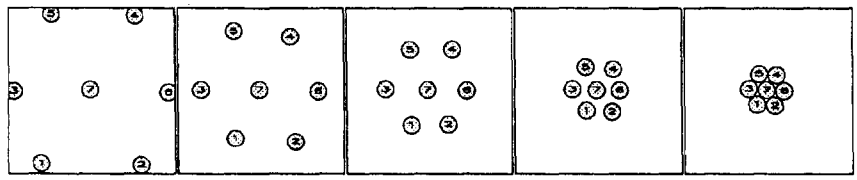

Figure 5: Five 7-Ball Assemblies of Varying Difficulty: (from left to right) (a) $\beta(d) = 10^{-2}$, (b) $\beta(d) = 10^{-9}$, (c) $\beta(d) = 10^{-16}$, (d) $\beta(d) = 10^{-28}$, (d) $\beta(d) = 10^{-44}$.

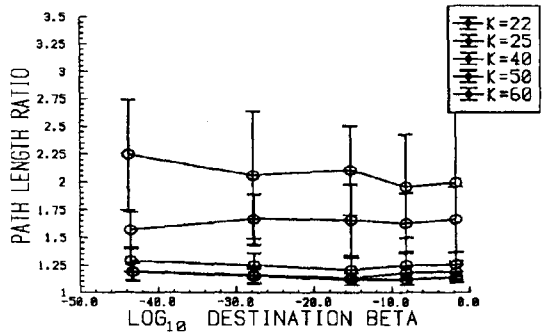

Figure 7: Path Length Ratio vs $\beta(d)$ for each of the five 7-ball assemblies of Figure 5 for Five k Values. Each data point represents the Mean and Standard Deviation of 25 runs with random initial configuration.

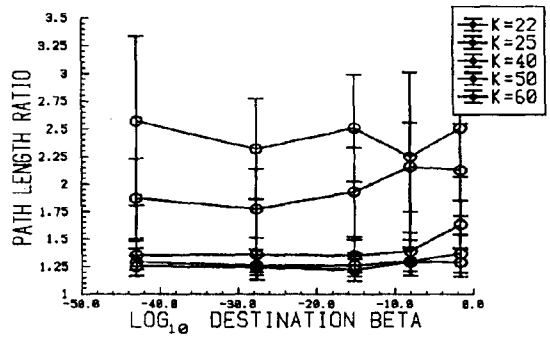

Figure 8: Switched Algorithm: Path Length Ratio vs $\beta(d)$ for each of the five 7-ball assemblies of Figure 5 for Five k Values. Each data point represents the Mean and Standard Deviation of 25 runs with random initial configuration.

adverse cases previously described[5].

We conclude that the path-length performance of the gradient algorithm is typically no worse than twice the corresponding euclidean distance, and thus no more than twice the optimal length solution as well. Moreover, this surprisingly good performance is maintained even in the face of a variety of difficult (closely-packed) assemblies.

5.3 Performance of the Switched Assembly Algorithm

We have constructed a numerical simulation to investigate the performance of the switched assembly algorithm presented in Section 4.2. Figure 8 shows the means and standard deviations of the switched algorithm's path length ratios[6], each for 25 runs, at five different $\beta(d)$ values and five different values of the parameter k. This figure represents the switched version of the data portrayed in Figure 7. A comparison of Figure 7 (unswitched) and Figure 8 (switched) reveals essentially similar performance, with the switched algorithm pathlength ratio only 10% greater than that of the unswitched.

Figure 9 shows the corresponding mean and standard deviations for the *number of switches* for the runs of Figure

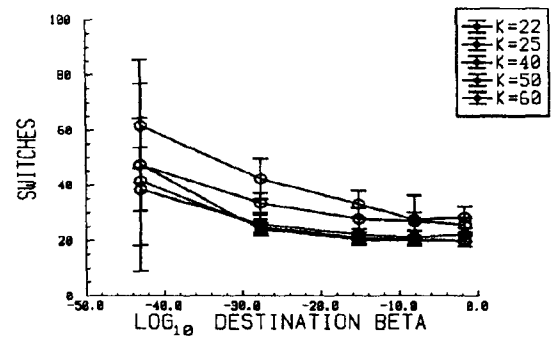

Figure 9: Switched Algorithm: Switch Count vs $\beta(d)$ for each of the five 7-ball assemblies of Figure 5 for Five k Values. Each data point represents the Mean and Standard Deviation of 25 runs with random initial configuration.

[5] In particular, the number of switches when the sequential version of this algorithm was run in these cases was not statistically different from those involving less adverse initial conditions.

[6] Note that the path-length ratio for the unswitched algorithm is with respect to the configuration space euclidean norm from initial to final configuration, whereas the the switched version is computed with respect to the sum of the individual ball distances.

8. Here we observe that the number of switches required to complete an assembly rises perceptibly as a function of the assembly difficulty. The "easy" assemblies here require each ball to be switched (grasped, moved, and released) only about three times, while the "difficult" assemblies have both a higher mean number of switches as well as substantially higher variance.

Since a human can easily perform these assembly tasks with only one or two switches (grasp, move, release) per ball, the switching algorithm is obviously sub-optimal. It offers the useful possibility, however, of *automatically* planning the manipulation of an assembly with a great many degrees of freedom with a robot of just a few degrees of freedom. Moreover, while this (admittedly sub-optimal) algorithm fails to compare with human performance in this simple example, we hope this approach will provide an automatic and constructive technique for assemblies of much higher complexity for which intuition may fail.

6 Conclusion

We have presented an approach to assembly motion planning that holds the promise of providing as well a control scheme capable of effecting the physical motions needed to implement the plan. We have explored the performance of a concrete instance of this approach in a very simple problem setting — re-arranging disks on a plane into specified finished patterns. Simulations suggest that our algorithms yield configuration space paths that are *typically* (in a statistical sense) better than within a factor of two of the optimal length. Performance varies with a design parameter (k) whose proper adjustment can apparently deliver configuration space paths that are typically within 25% of the optimal length. Somewhat surprisingly, performance appears to vary little with assembly "difficulty".

Motion sequences for one manipulator commanded to build an assembly involving n separate bodies have path lengths roughly 10% higher than those corresponding to simultaneous motion of all n bodies. The number of "switches" in these cases seems to increase with assembly "difficulty" and to decrease with increasing k. In an n body assembly plan sequence for a single robot, the number of switches in the cases of even greatest "difficulty" is typically less than n per body.

As with any exact solution to a task that involves a version of the generalized piano-movers problem, it must be expected that the computational complexity of our approaches will increase exponentially in the degrees of freedom — in our case, the number of pieces to be assembled. We have found that these algorithms run reasonably quickly when the number of pieces does not exceed ten , but that further increases cause increasingly significant delays (for example, run times of roughly 5 min. on a Sun SPARCstation-1 for a fifteen piece problem, the largest we have run to date). In *practice*, the salient upper limit on number of pieces appears to be dictated by the numerical precision of floating point representation employed. The matter deserves careful attention.

Since the foundation for these algorithms rests upon our earlier work with Navigation Functions [5], there is good reason to hope that the trivial geometry of our present task domain hides neither unforeseen computational complexity nor unresolvable constructive road blocks when the shape of the pieces is generalized. Rimon and Koditschek have shown how to deform simple problems into all topologically equivalent relatives (no matter how geometrically intricate) in a very simple topology [7]. Rimon has begun to suggest how to extend the constructions explored here to the topology resulting from pieces that are not spherically symmetric [8]. We are currently completing a proof which demonstrates that (6) is a Navigation Function.

We hope to construct a working laboratory implementation of these ideas in the near future.

Acknowledgments

We thank Dr. Elon Rimon for a number of valuable discussions concerning this problem.

References

[1] L. S. Homem de Mello and A. C. Sanderson. A correct and complete algorithm for the generation of mechanical assembly sequences. *IEEE Transactions on Robotics and Automation*, 7(2):228–240, April 1991.

[2] D. E. Koditschek. An approach to autonomous robot assembly. *Robotica*, (submitted), 1991.

[3] Daniel E. Koditschek. Adaptive techniques for mechanical systems. In *Fifth Yale Workshop on Applications of Adaptive Systems Theory*, pages 259–265, New Haven, CT, May 1987. Center for Systems Science, Yale University.

[4] Daniel E. Koditschek. The control of natural motion in mechanical systems. *ASME Journal of Dynamics Systems and Measurement*, Dec 1991.

[5] Daniel E. Koditschek and Elon Rimon. Robot navigation functions on manifolds with boundary. *Advances in Applied Mathematics*, 11:412–442, 1990.

[6] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, Cambridge, UK, 1988.

[7] E. Rimon and D. E. Koditschek. The construction of analytic diffeomorphisms for exact robot navigation on star worlds. *Transactions of the American Mathematical Society*, 327(1):71–115, Sep 1991.

[8] Elon Rimon. A navigation function for a simple rigid body. In *IEEE International Conference on Robotics and Automation*, pages 546–551, Sacramento, CA, USA, 1991.