Proceedings of the 2001 IEEE
International Conference on Robotics & Automation
Seoul, Korea • May 21-26, 2001

Programming by Demonstration: Removing Suboptimal Actions in a Partially Known Configuration Space

J. R. Chen
Department of Engineering, FEIT
The Australian National University
Canberra, ACT, 0200, Australia
E-mail: chen@faceng.anu.edu.au

A.Zelinsky
Department of Systems Engineering, RSISE
The Australian National University
Canberra, ACT, 0200, Australia
E-mail: Alex.Zelinsky@anu.edu.au

Abstract

Programming by demonstration is a promising approach to automatic robot programming, however methods are required to remove suboptimal actions that can be demonstrated by end users. In this paper we use the partial knowledge of Configuration Space (C-space) derived in previous work to remove suboptimal actions from a demonstration. Our idea is to use demonstrated paths to predict what regions in C-space are obstacle free. Suboptimal actions in a demonstration are then avoided by planning alternative actions that pass through the obstacle free regions. Experimental results show the validity of the approach. A demonstrated path containing significant sub-optimality was converted by the approach into a short, efficient path suitable for execution by the robot.

1 Introduction

Recently there has been growing interest in the field of service robotics, where robots are utilized for tasks in a domestic environment. A major question in service robotics regards end user programming. A typical householder will not know how to program a robot in the usual way, ie. by writing computer code. A new programming method is required that provides a more natural programming interface for non-technical users. A promising solution is Programming by Demonstration (PbD). Here, the end user demonstrates the task to be programmed. A PbD interface then interprets the demonstration and determines the control details required by the robot to achieve the task.

A well known problem in PbD is that the demonstration can contain erroneous or suboptimal actions, ie. "noise" [2, 10, 4, 8]. For example, Delson and West [2] identify that, in a pick and place task through a field of obstacles, a human will naturally introduce noise into the demonstration by using different paths to traverse regions were the gap between obstacles is large. De Schutter et al [8] found that a demonstration of a peg-in-hole task could contain actions by the demonstrator that were suboptimal, erroneous, or even unintended. Clearly, having the robot directly copy the demonstration will not be optimal. The solution is to identify and remove any noise from demonstrated paths before they are programmed into the robot.

In this paper we present a new approach to removing noise from a demonstration. Our work here follows on from previous work [3], where we derived the configuration space (C-space [5]) of a task from demonstration. Our idea then is for noise removal in two steps, (i) generate C-space for the task from demonstration, and (ii) use the C-space information to remove noise from demonstrated paths. We focus in this paper on presenting a solution for step (ii). Note that work in [3] provides only a partial knowledge of C-space, ie. this work only derived regions of C-space that were visited in the demonstration. As such, well known path planning methods [5] cannot be applied to derive a noise free path.

Previous approaches to noise removal in PbD include [2, 4, 8]. Kaiser and Dillmann, and De Schutter et. al. [4, 8] present approaches that use thresholding, smoothing, and loop removal techniques. For example, Kaiser and Dillmann [4] propose two types of noise removal for a peg-in-hole task. First, they remove ineffective actions, ie. actions that changed the configuration of the peg by less than a predefined threshold. Second, they remove actions that were later corrected, ie. those that were partly or fully negated at following time steps. This is the process of removing loops from the demonstrated path. Work in [4, 8] has been successful in removing obvious flaws in the demonstration, however it is limited to deriving paths that contain only demonstrated points. As such, less obvious flaws cannot be removed, eg. corrected actions that do not form explicit loops in the path. Delson and West [2] propose a different approach. They address the case where all demonstrated paths start and end at the same point. They identify an obstacle-free en-

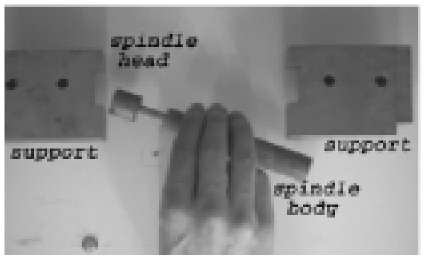

Figure 1: The spindle insertion task chosen for PbD

velope formed by the outer-most lying paths that were demonstrated. A noise-free path is then constructed to lie completely within the demonstration envelope. This approach can derive paths that contain points that were not demonstrated. As such, it is capable of removing sub-optimalities from the demonstrated path that [4, 8] cannot. However, the approach is restricted to finding paths that lie within the envelope. This can be a limitation, for example, where only one demonstration is provided. In addition, it has been presented for C-space of dimension 2 or 3. The method is not easily extendable to higher dimensions.

Our approach to noise removal compares well to previous approaches. The approach can derive paths that contain undemonstrated points. As such, it can remove a greater range of sub-optimalities than [4, 8]. Compared to [2] our approach has the advantages that (a) it does not assume that all demonstrations pass between the same start and end points, (b) it can derive paths that lie outside the demonstration envelope, and (c) it can be applied to find paths in C-space of any dimension. A limitation of our approach compared to [2] is that caution needs to be applied when tuning the parameters of the method, otherwise it may produce a non-obstacle free path.

2 Problem Formulation

Our aim is to present noise removal for a demonstration of a typical household task. The task chosen is shown in Figure 1. It is based on the domestic chore of changing rolls on a paper roll holder, and involves inserting an axially compressible spindle between two supports. The task involves four degrees of freedom, three to describe the position/orientation of the spindle body relative to the supports(y,z,θ), and one to describe the compression of the spindle head relative to the spindle body (δ).

We model our task as a Hybrid Dynamic System (HDS). The spindle insertion task is in essence an assembly task, involving contact and constrained motion between task objects. Hybrid Dynamic Systems have been presented as a good way to model assembly tasks [7]. In its most general form, a HDS involves a continuous-time system interacting with a discrete-event system [9]. For assembly, the continuous-time system represents the continuous-time dynamics of the spindle. That is, as (i) a differential equation describing the free-space motion of the spindle relative to the supports, and (ii) a set of constraint equations describing the constraint on spindle motion when the spindle is in contact with the supports. In contrast, the discrete-event system captures the *discrete* nature of the assembly dynamics. It describes the assembly as a sequence of asynchronous *discrete events* occurring through time. A discrete event is defined to occur when the set of constraints on the spindle motion changes. Each distinct constraint set possible in the task is defined as a *discrete state*. Generally a discrete state will correspond to a unique contact formation between the spindle and supports. We show in Figure 2, six sequences of discrete states that were demonstrated for the task. The figure shows how an assembly sequence is nicely described as a sequence of discrete states in the HDS. To make referencing easier, we give each state a *state number*, eg. state 2 is the start state, state 1 is the goal state, etc. Note that the six paths shown in Figure 2 form the demonstration set we used to derive C-space in [3].

The overall problem to be solved involves noise removal from a complete demonstrated path. Then HDS modeling simplifies the problem to be solved. Rather than be concerned with noise removal over the entire path, we can address the problem separately in each state. To formulate this problem exactly, we must first understand something of the topology of C-space [1], and its relationship with our definition of a state in the HDS. C-space consists of an obstacle free region (C_{free}), an obstacle defining region, and a region defining the boundary between the two ($C_{contact}$). Then the no-contact state in the HDS (state 2) corresponds to C_{free}. That is, any spindle configuration in state 2 will correspond to a point in C_{free}. All other states in the HDS involve contact between the spindle and supports, and so together correspond to $C_{contact}$. Individually, they each define a *C-surface*, a patch of curved surface that defines part of $C_{contact}$. Each demonstration in Figure 2 defines a path through C-space. Clearly the path will visit the C-surface of each state in the demonstration. Let γ_i be the i^{th} distinct state in the demonstration, and let c_i be its C-surface. Then we know that some segment of the path will exist on c_i. Call this segment Π. Then, the

[1]see [5] for a detailed presentation of the topology of C-space

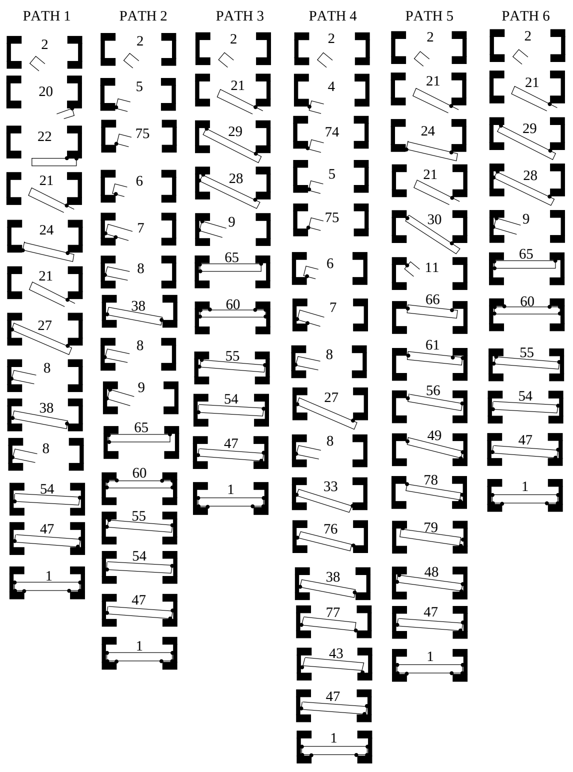

Figure 2: The demonstration set used to construct C-space

first requirement of our noise removal method is that it derive a noise-free path between the start and end points of Π.

We note that C-surfaces can be of different dimension. Let C-space have dimension n. Then states where the spindle loses one dof because of contact with the supports will have C-surfaces of dimension n–1. States involving a single spindle dof will have C-surfaces of dimension one. We want our noise removal method to work for any state in the task. That is, our second requirement on the method is that it cope with deriving paths on C-surfaces ranging in dimension from 1 to $n-1$. In addition, it must be able to determine a noise free path in C_{free} (which is of dimension n).

Recall that $C_{contact}$ is defined by a set of intersecting C-surfaces. The extent of a C-surface c_i in $C_{contact}$ is defined by where it intersects with its neighboring C-surfaces. Neighboring C-surfaces to c_i belong to states that were demonstrated immediately before or after γ_i. Neighboring C-surfaces can only have dimensions one greater or one less than c_i. This is because states demonstrated immediately before or after γ_i correspond to the gain or loss of a single spindle dof compared to γ_i. Let c_i have dimension k. Then neighboring C-surfaces to c_i can be divided into those

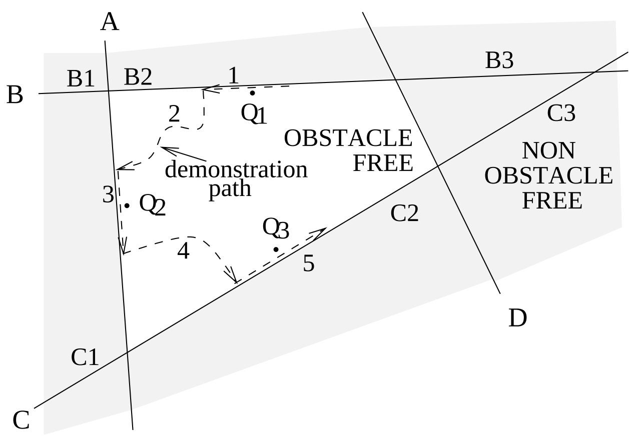

Figure 3: Demonstration segments identify regions on a C-surface that are likely to be obstacle free

of dimension k+1, and those of dimension k–1. Neighboring C-surfaces in general define obstacles on c_i that we wish to avoid. If we generate a path lying exactly on c_i, then we are guaranteed to avoid neighboring C-surfaces of dimension $k+1$. That is, the third requirement for our noise removal method is that it generate paths exactly on the C-surface c_i. We denote as a *boundary* on c_i, each neighboring C-surface to c_i of dimension k–1. Then the fourth and final requirement on our noise removal method is that it generate paths that lie within all boundaries on c_i. We now present a method that forfills our four requirements.

3 Removing Noise from Demonstrated Segment Π

Since we do not have full knowledge of C-space, not all boundaries on c_i are guaranteed to be known. We show in Figure 3 a 2-D C-surface where boundaries A,B and C were visited in the demonstration and are known, while D is unknown. The figure shows that some portion of a known boundary may exist behind an unknown boundary, eg. C3 and B3, and hence does not really divide an obstacle free region from an obstacle defining region. We note that a boundary is guaranteed to divide obstacle-defining and obstacle-free regions along a segment traversed in the demonstration, eg. segments 1, 3, 5. We call such segments, *boundary segments*. We observe that if the C-surface is of finite size (as is usually the case), then a point immediately in front of the boundary segment will be obstacle free, eg. points Q_1, Q_2, and Q_3. We use this observation as the basis for growing obstacle free regions on the C-surface. We grow a free region in front of each boundary segment of the C-surface. If the region is grown very small, then we are guaranteed that it will be obstacle free. However a small region is of limited use for path planning purposes. Hence we grow a region of useful size and accept that the region will only likely be obstacle free. We call such a region a *likely free region*. Once a likely free region is identified, we use a road-map type approach to path planning, similar to those presented in [6]. We randomly generate points within the region, and use a simple path planner to create a connectivity graph $\mathcal{L}$ that records which points have an obstacle free path between them. Apart from points in likely free regions, we also know that points in demonstrated paths interior to known boundaries are obstacle free, eg. points in segments 2 and 4. We call such segments *interior segments*. We use the same simple path planner to create a graph $\mathcal{D}$ that records the connectivity between points in different interior segments. Finally we combine graphs $\mathcal{L}$ and $\mathcal{D}$ into a graph $\mathcal{K}$ that represents the connectivity of all obstacle free points on the C-surface. We identify the nodes in $\mathcal{K}$ that represent the start and end points of Π. We search for the minimum cost path between these nodes to give the final noise-free path. The process can be divided into four distinct steps. They are:

- Creating boundary segments
- Growing likely free regions
- Creating interior segments
- Creating a connectivity graph $\mathcal{K}$, and searching $\mathcal{K}$ for our final noise-free path

We present the details of each step in the following four sub-sections.

3.1 Creating Boundary Segments

Three steps are required to create boundary segments for γ_i, (i) finding boundary states to γ_i, (ii) identifying *raw* boundary segments for the boundary states found in (i), and (iii) projecting points in raw boundary segments to give a set of *clean* boundary segments. To achieve (i), recall that a state γ_i is defined in our HDS by a set of constraints on spindle motion. Let Ω_i be the set of constraints that define γ_i. Then we choose a *boundary state* of γ_i as any other demonstrated state that is defined by the set of constraints Ω_{bnd}, where $\Omega_{bnd} \supset \Omega_i$. That is, constraints present in γ_i will also be present in its boundary states. Step (ii) is then straightforward. We select as a raw boundary segment, any path demonstrated in the boundary states of γ_i. Step (iii) is required because points in raw boundary segments will not generally lie exactly on the C-surface of the boundary state. That is, the regression analysis of [3] derived C-surface equations for each state that *best-fit* the raw demonstration data. We create clean boundary segments by orthogonally projecting [1] all points in a raw boundary segment onto the C-surface of the boundary state. From now on, we refer to clean boundary segments simply as *boundary segments*.

3.2 Growing Likely Free Regions

We grow a likely free region by generating a region of points on our C-surface c_i immediately in front of a boundary segment. Each point in the region is determined as follows. First, a point P in the boundary segment is randomly selected. Next, a distance value dst[2] is randomly chosen using the uniform probability distribution over the interval $(0, \mathsf{md}]$. Parameter md denotes a maximum distance value, and determines how big the likely region is grown. A point in the likely free region is then determined by generating a point Q that (a) lies a distance dst from P, and (b) lies on our C-surface c_i. In addition to (a) and (b), point Q must also satisfy the condition (c) that it lie within known C-surface boundaries. We refer to the region on c_i lying within known boundaries as the *bounded region* for c_i. For example, we present in Figure 4 a 2-D C-surface with six boundaries c_1 to c_6, each described by equations $\phi_1 = 0$ to $\phi_6 = 0$ respectively. Here, the bounded region consists of the union of regions labelled 1, 2, and 3. The remainder of this section presents the details of how we ensure that Q lies inside the bounded region of a C-surface c_i.

Let $\phi_{bnd} = 0$ be the equation of the C-surface for γ_{bnd}, a boundary state to γ_i. That is, equation $\phi_{bnd} = 0$ defines a known boundary on our C-surface c_i. Then we identify that any Q we generate will satisfy one of the following conditions:

$$-\mathsf{eps} < \phi_{bnd}|_Q < \mathsf{eps} \tag{1}$$

$$\phi_{bnd}|_Q \leq -\mathsf{eps} \tag{2}$$

$$\phi_{bnd}|_Q \geq \mathsf{eps} \tag{3}$$

where $|_Q$ denotes the equation evaluated at point Q, and eps is a parameter of small value. If (1) is satisfied, then Q lies on, or very close to the boundary, while if (2) or (3) are satisfied, Q lies to one side of the boundary. We are never interested in Q's that satisfy

[2]throughout this paper, symbols in plain upright text denote parameters of our path derivation method

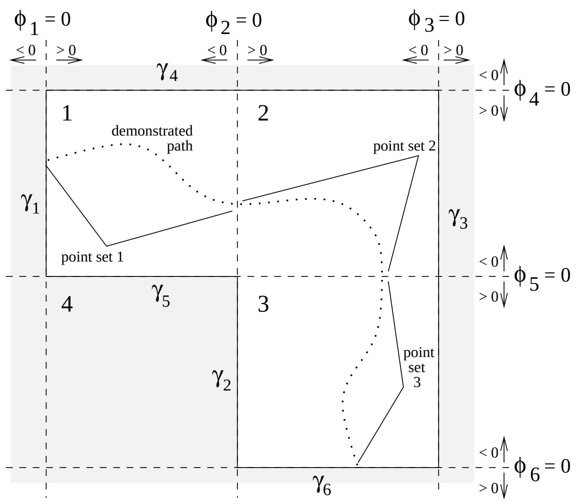

Figure 4: Example of how demonstrated points determine a set of valid bounded sub-regions

(1). A Q lying exactly on the boundary is not obstacle free, and a Q lying very close to the boundary may not be obstacle free (recall that the equations we derived for C-surfaces in [3] are best-estimates that contain some error). Rather, we are interested in generating Q that satisfy either (2) or (3). Note that the choice of whether we want Q to satisfy (2) or (3) will depend on which side of the boundary is obstacle free.

In general, there will be n_b known boundaries to our C-surface c_i. Denote the equations of boundaries 1 to n_b respectively as:

$$\phi_{bnd\_1} = 0, \ \ \phi_{bnd\_2} = 0, \ \ \ldots \ \ , \phi_{bnd\_n_b} = 0 \qquad (4)$$

We note that a Q which satisfies one or more equations in (4) will lie on a boundary of the bounded region. We saw that we are not interested in such Q. Rather, we want Q that lie on the obstacle free side of all boundaries by at least a distance eps. That is, we must form a set of inequalities by *casting* each equation in (4) to be an inequality of the form (2) or (3). Call such a set of inequalities a *boundary inequality set.* Then we note that a boundary inequality set defines a region on our C-surface c_i. For example, one boundary inequality set for the boundary equations $\phi_1 = 0$ to $\phi_6 = 0$ of our example in Figure 4 would be:

$$\begin{array}{lll} \phi_1 \geq \mathsf{eps}, & \phi_2 \leq -\mathsf{eps}, & \phi_3 \leq -\mathsf{eps}, \\ \phi_4 \geq \mathsf{eps}, & \phi_5 \leq -\mathsf{eps}, & \phi_6 \leq -\mathsf{eps} \end{array} \qquad (5)$$

Then the region on the 2-D C-surface in Figure 4 that is defined by the boundary inequality set (5) is region 1. That is, a point Q is guaranteed to lie in region 1 if it satisfies all inequalities in (5). We note that in general the bounded region on c_i cannot be specified by a single boundary inequality set. Denote as a *bounded sub-region* the region defined by a single boundary inequality set, ie. region 1 in Figure 4 is a bounded sub-region. Then, we identify that the bounded region of a C-surface can be specified as a union of bounded sub-regions. For example, the bounded region in Figure 4 is given by the union of bounded sub-regions 1, 2, and 3. A point Q can then be tested to see if it lies within the bounded region by testing to see if it lies within any of its component bounded sub-regions. The question is then one of how to select the set of bounded sub-regions that make up our bounded region. Note that not all bounded sub-regions that can be generated for a certain set of boundary equations will be obstacle free. For example, region 4 in Figure 4 is a valid bounded sub-region, however region 4 defines an obstacle.

Our solution is to use the set of demonstration points on the C-surface to determine what bounded sub-regions are valid. Let μ be a possible bounded sub-region for c_i, and Υ be the boundary inequality set that defines μ. Then we say that μ is a valid bounded sub-region if there exists a point in the demonstration set on c_i which satisfies all inequalities in Υ. For example, in Figure 4 the demonstrated points in point-set 1 show region 1 to be a valid bounded sub-region because they satisfy all inequalities in (5). Similarly, demonstration point sets 2 and 3 show regions 2 and 3 as valid bounded sub-regions. Region 4 is not included as a valid bounded sub-region because it does not contain any demonstrated points. This approach is a conservative solution because we may miss some valid bounded sub-regions in which no demonstrated points exist. However, we take this approach because it guarantees that we do not accept Q lying in bounded sub-regions that define obstacles.

3.3 Creating Interior Segments

We denote as a *raw interior segment* a path demonstrated in state γ_i. We use the term *interior* because we know that such segments lie within known boundaries of c_i. Note that Π is one of the raw interior segments on c_i. Our aim here is to derive *clean* interior segments. There are two requirements on clean interior segments to which raw interior segments do not comply. These are (i) points in raw interior segments do not lie exactly on c_i (ie. work in [3] derived equations that best fit data points), and (ii) the start and end points in raw interior segments do not lie exactly on the boundaries of c_i. We achieve (i) by orthogonally projecting all points in the raw interior segment onto c_i. For (ii), let η be an arbitrary interior segment on c_i. Denote as γ_{ent} and γ_{ex} the previous and following states to γ_i when η was demonstrated. Then we note that the start and end points of η will lie close to the C-surfaces of γ_{ent} and γ_{ex} respectively. We ensure that they lie exactly on these C-surfaces by orthogonally projecting the start point in η onto the C-surface of γ_{ent}, and its end point onto γ_{ent}. A set of clean interior segments is achieved by repeating steps (i) and (ii) for all raw interior segments on c_i. Denote the clean interior segment corresponding to Π as $\acute{\Pi}$. From now on, we refer to clean interior segments simply as *interior segments.*

3.4 Creating a Connectivity Graph

The previous two sections have been devoted to generating points on our C-surface c_i. Two types of points were generated, points in likely free regions, and points in interior segments. We show in Figure 5 a possible outcome of the point generation process for a simple 2-D planar c_i. It shows three likely free regions (labelled A, B, and C) generated in the bounded region on c_i. In addition it shows two interior segments (labelled η_1 and η_2) that were also generated. Our aim in this section is to create a graph $\mathcal{K}$ that represents the connectivity between all generated points on c_i. Such a graph should have a node to represent each generated point. It should have arcs existing between nodes whose points are connected by an obstacle free path. In addition, we assign to each arc in the graph a value equal to the cost of traversing the obstacle free path it represents. We construct this graph $\mathcal{K}$ in three steps, (i) create a graph $\mathcal{L}$ representing the connectivity between points in likely free regions, (ii) create a graph $\mathcal{D}$ representing the connectivity between points in interior segments, and (iii) combine graphs $\mathcal{L}$ and $\mathcal{D}$ into $\mathcal{K}$.

Creating $\mathcal{L}$: We first create a point set consisting of all points in all likely free regions on c_i. Then a node in $\mathcal{L}$ is created for each point in the point set. To construct arcs in $\mathcal{L}$ we must determine two things for each pair of points in the point set, (a) if the points are connected, and (b) the cost of traversing between connected points. We say that two points Q_1 and Q_2 in the point set are connected if two conditions are satisfied. First, that the straight line path between Q_1 and Q_2 is obstacle free, ie. that every point on the straight line lies within the bounded region of the C-surface. For example, points Q1 and Q2 in Figure 5

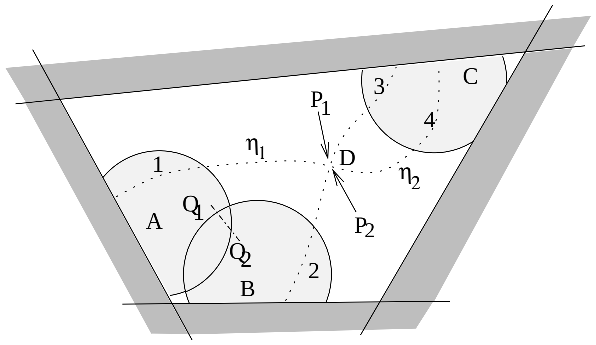

Figure 5: Example of points generated by our method for a simple, planar c_i

have a straight line path between them that lies fully inside the bounded region on c_i. Second, that the Euclidean distance between Q_1 and Q_2 does not exceed a maximum connected distance parameter mcd_l. We apply the second condition for the following reasons. First, recall that our point set consists of points from distinct likely free regions. We do not want to connect points in likely free regions that lie far apart ,eg. between points in likely free regions A and C in Figure 5. That way we are less likely to connect points on c_i between which an unknown boundary exists. Second, intermediate points on the straight line between Q_1 and Q_2 will not in general lie exactly on the C-surface due to its curvature. However, if Q_1 and Q_2 do not lie too far apart, then intermediate points will lie close enough to the C-surface for the purpose of checking whether they are obstacle free. Third, applying such a condition is advantageous from a computational point of view. The number of points to be tested for connection increases rapidly as the allowed distance between the points increases. In addition, note that testing for connectivity only between points lying a distance less than mcd_l apart does not detract from the end performance of our method. That is, points lying far apart that really should be connected, (eg. those existing in the same, or overlapping, likely free regions), will be connected efficiently at the final path planning stage. In general they will be connected by a compound path passing through a sequence of intermediate points, each separated by a distance of less then mcd_l.

Our second requirement for creating the arcs in $\mathcal{L}$ was (b) to determine their cost. We denote as ϖ_1 and ϖ_2 the nodes in $\mathcal{D}$ that represent points Q_1 and Q_2, and as ϑ the arc in $\mathcal{D}$ that connects ϖ_1 and ϖ_2. Then we calculate the cost of ϑ as the Euclidean distance between Q_1 and Q_2. Euclidean distance is the appropriate measure of cost here, since we desire that a minimum cost path in the final connectivity graph $\mathcal{K}$ represents the shortest distance path on our C-surface.

Creating $\mathcal{D}$: We create $\mathcal{D}$ in two steps, (a) create a distinct connectivity graph for each interior segment on c_i (eg. distinct connectivity graphs for η_1 and η_2 in Figure 5), and (b) combine graphs constructed in (a) into $\mathcal{D}$. Step (a) is straight-forward because the connectivity of points in any interior segment is known, ie. apart from the start and end points, each point is connected to two other points, a previous point, and a following point. We create a graph for each interior segment with nodes and arcs that reflect this connectivity. We assign as costs to the arcs in the graph the Euclidean distance between each of the sequential point pairs.

In step (b) we must combine the graphs created in step (a) into a single graph $\mathcal{D}$. The process of combining these graphs means deciding if and where interior segments on c_i *intersect*, That is, we should create a connecting arc between two graphs $\mathcal{D}_1$ and $\mathcal{D}_2$ derived in step (a) when the interior segments η_1 and η_2 represented by these graphs are found to intersect, eg. at point D in Figure 5. We say that η_1 and η_2 intersect at points P_1 (in η_1) and P_2 (in η_2), when the Euclidean distance between P_1 and P_2 is less than a parameter mcd_d. If this is the case, an arc is connected between the node in $\mathcal{D}_1$ that represents P_1 and the node in $\mathcal{D}_2$ that represents P_2. Then $\mathcal{D}$ is created by repeating this process for every possible P_1 and P_2 in our set of interior segments on c_i.

Creating and Searching $\mathcal{K}$: We create $\mathcal{K}$ by combining graphs $\mathcal{L}$ and $\mathcal{D}$. The idea is to connect points in interior segments with those in likely free regions where an interior segment passes through a likely free region (eg. along segments labelled 1,2,3 and 4 in Figure 5). We create a $\mathcal{K}$ that represents such connectivity as follows. For each point P in each interior segment η, find a point Q in any likely free region that lies within a distance mcd_l away. If such Q's exist, then for each Q found, create an arc between the node that represents point P in $\mathcal{D}$, and the node that represents point Q in $\mathcal{L}$. Set the cost of the arc to the Euclidean distance between the points P and Q. Once $\mathcal{K}$ is determined, we can achieve a noise free path between the start and end points of our cleaned demonstrated segment $\acute{\Pi}$. It is achieved by searching in $\mathcal{K}$ for the minimum cost path between the nodes that represent the start and end points of $\acute{\Pi}$.

We have introduced a number of parameters for our method, eg. md, eps, mcd_l, etc. A key issue in ensuring good performance from our method is of setting these parameters to appropriate values. This is especially true for parameter md. Recall that the value of md determines how big a likely free region is grown. A balance must be struck in setting md between the benefit of large likely free regions for path generation purposes, and the risk of generating points that lie on the non-obstacle-free side of unknown boundaries. At this stage in the research we are still investigating methods for automatically setting parameters to appropriate values. For the results presented in the next section, the values of parameters were set manually.

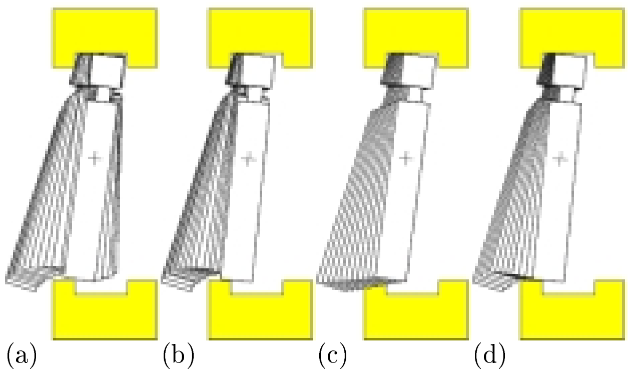

Figure 6: (a) original demonstrated path containing noise, (b) the noise-free path derived by our method, (c) the shortest length path, and (d) a path that passes too close to obstacles

4 Results

We show in Figure 6(a) a path used by the demonstrator to traverse state 8 between states 7 and 38. We show this path in task-space rather than C-space due to the difficulties involved with presenting a 3-D hyper-surface graphically (the C-surface corresponding to state 8 is a 3-D hyper-surface). Each spindle configuration in Figure 6(a) corresponds to a point in the path on the C-surface. Notice then how this demonstrated path contains significant noise. The demonstrator has used an overly long path to traverse between the two states. The path resulted because the demonstrator became confused about the position of the spindle relative to the supports. His initial aim was to pass into state 65 (see Figure 2). However on not finding this state, his reaction was to retract the spindle back towards himself, resulting in state 38.

We applied our noise removal method to this path. The resulting noise-free path is shown in Figure 6(b). There are three main things to note about the noise-free path. First, it is significantly shorter than the demonstrated path. It passes efficiently between the start point in state 7 and end point in state 38. Second, it avoids obstacles, ie. it avoids unwanted contacts between the spindle and supports that would see the process accidently move into states neighboring state 8 that are not state 38, eg. state 27, 9, or 33, etc. (see Figure 2). For example, we show in Figure 6(c) the path resulting if the method had simply selected the shortest path (ie. straight line path in C-space) between the start point in state 7 and the end point in state 38. This path does not avoid obstacles, and would have resulted in the assembly process accidently moving into state 27. Paths that avoid obstacles are selected by the method because we ensured that points in both likely free regions and interior segments were generated within the bounded region on the C-surface. That is, the path in Figure 6(c) was not selected because it contains points lying outside the bounded region of the C-surface of state 8. The third thing to note about our noise-free path in Figure 6(b) is that it maintains a safe distance from obstacles. For example, a path has not been derived where the the bottom of the spindle "scrapes" along the top of the lower support (shown in Figure 6(d)), even though this path has a shorter length in C-space than the one derived. This is a result of ensuring that points in likely free regions lie a minimum distance of eps away from boundaries.

The performance of our approach for noise removal compares well with previous approaches. We have derived a noise free path that contained undemonstrated points. As such it was able to remove an action that was later negated, even though an explicit loop was not formed in the path. In addition, we have presented experiments where the demonstrated paths in a state did not all pass between the same start and end points. That is, our method produced a noise free path without the existence of an explicit demonstration envelope to define an obstacle free region. The path was derived on a C-surface of dimension three, however clearly it can applied to finding a path on C-surfaces of dimension greater than three.

5 Conclusion

Facilitation of end user programming is one of the main obstacles for success in service robotics. Programming by Demonstration provides a promising solution, however methods are required to cope with noisy and suboptimal actions that are demonstrated by end users. We proposed that such actions can be avoided by (i) deriving a C-space representation of the task from demonstration, then (ii) using C-space information to remove noise from demonstrated paths. In this paper we concentrated on solving problem (ii). We used HDS modeling to convert the noise removal problem for the complete demonstrated path into a noise removal problem on the C-surface of each state in the path. Our approach identified two types of obstacle free points on the C-surface. First, points that were visited in the demonstration. Second, points in likely free regions. We represented the connectivity of these points using a connectivity graph. A noise-free path between any two points on the C-surface could then be obtained by searching for the minimum cost path in the connectivity graph. We successfully applied the approach on a demonstrated path that contained significant noise. We showed that our method could remove the suboptimal parts of the demonstrated path to form an efficient, noise-free path to be passed onto the robot for execution.

References

[1] D.C.Lay. *Linear Algebra and its Applications.* Addison Wesley, 1994.

[2] Nathan Delson and Harry West. Robot programming by human demonstration: Adaptation and inconsistency in constrained motion. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, 1996.

[3] J.R.Chen and B.J.McCarragher. Configuration space generation for assembly tasks from demonstration. In *Proceedings of Mechatronics and Vision in Practice*, September 2000.

[4] M. Kaiser and R. Dillman. Building elementary skills from human demonstration. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, pages 2700–2705, April 1996.

[5] Jean-Claude Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, 1991.

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

[7] Brenan J. McCarragher and Haruhiko Asada. The discrete event modelling and trajectory planning of robotic assembly tasks. *Journal of Dynamic Systems, Measurements and Control*, 117(3):394–400, October 1995.

[8] Wim Witvrouw Qi Wang, Joris De Schutter and Sean Graves. Derivation of compliant motion programs based on human demonstration. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2616–2621, April 1996.

[9] R.W.Brockett. Hybrid models for motion control systems. In H.L.Trentelman and J.C.Willems, editors, *Essays on Control: Perspectives in the Theory and Its Applications*, chapter 2, pages 29–5. Birkhauser, Boston, MA, 1993.

[10] Marjorie Skubic and Richard A. Volt. Learning force based assembly skills from human demonstration for execution in unstructured enviroments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1281–1288, May 1998.