

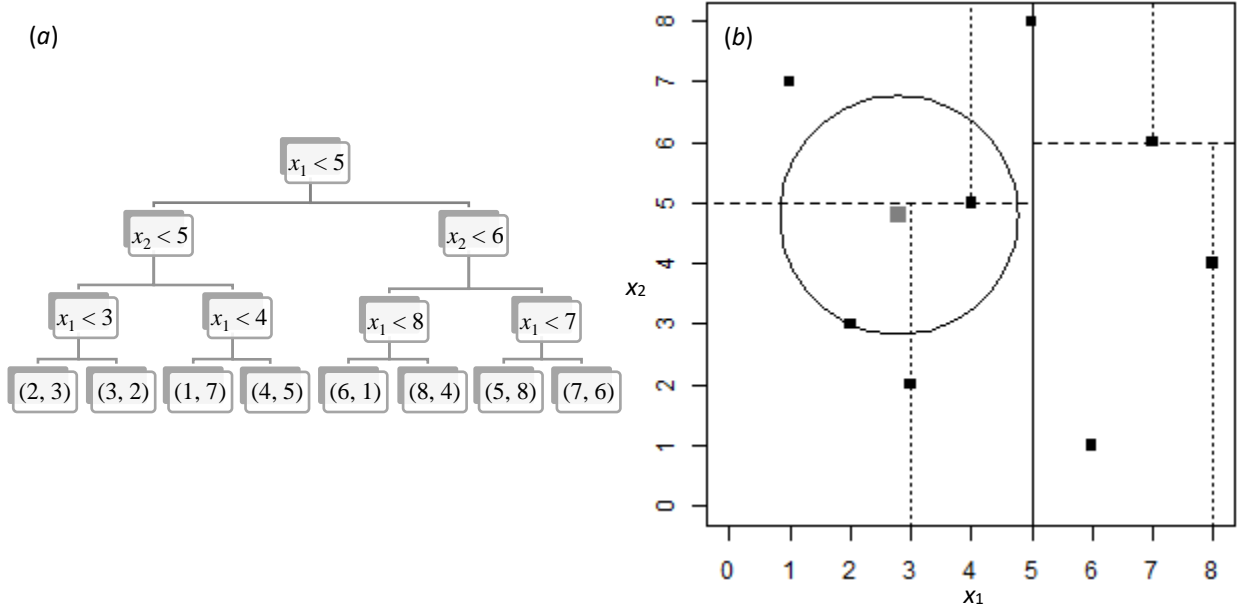
## 8. Supplementary Materials

### 8.1 *Kd*-Tree and Its Nearest Neighbor Search Algorithm

This section provides some descriptions of *kd*-tree and its nearest neighbor search algorithm. A *kd*-tree is a binary tree-like data structure, which was shown to be efficient in data storage and retrieval. Each *kd*-tree is constructed by two components. The first is nodes, which store groups of observations in the dataset. The root node contains the entire dataset. The second component is splits, each of which splits the observations in the corresponding nonterminal node into two children nodes based on a selected coordinate, i.e., into two half-spaces. Each of the terminal nodes contains a small group of observations, and is not split further as its children nodes' sizes would be less than some minimal node' size requirement. Unlike decision trees, which search for the optimal values to make splits, *kd*-trees conventionally make each split directly at the coordinate value of a selected observation (this process is known as point insertion). The splitting observations are usually selected as the ones having the median values of the splitting coordinate among the observations in the splitting nodes to produce a more balanced *kd*-tree, which has about the same depth on both sides of the root node.

An example of a balanced *kd*-tree for a 2D dataset with 8 observations is illustrated in Figure S1(a). The minimal terminal node's size is 1 in this illustration. Each node in the tree shows the splitting criterion, with the splitting value selected based on the median as mentioned above. The splitting coordinate is selected by cycling through each dimension as the tree is grown. The same coordinate is used for splitting at all nodes at the same level, i.e., having the same distance to the root node. In each split, observations with smaller values (of the splitting coordinate) than the splitting value are sent to the left side, and observations with larger (or equal) values are sent to the right side. Figure S1(b) shows the space partitions corresponding to the *kd*-tree in Figure S1(a). The larger, gray square in Figure S1(b) does not belong to the 2D dataset, but is a point to search neighbors for (discussed below) from among the 8 observations in the 2D dataset.

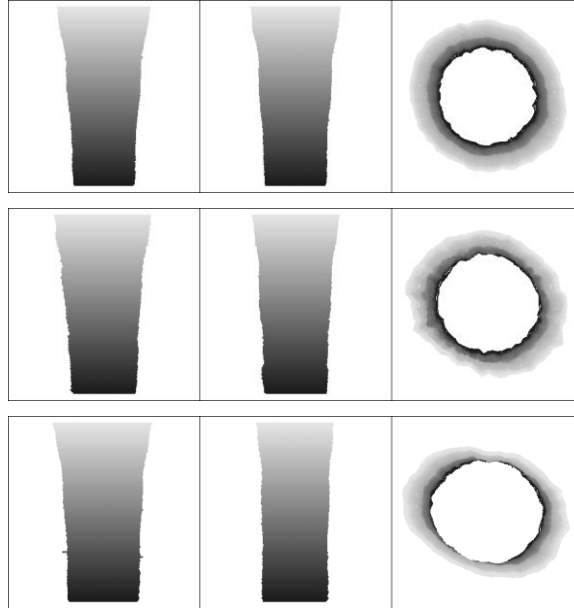
Given a new observation (e.g., the larger, gray square in Figure S1(b)), to find its  $K$  nearest neighbors from among a given dataset with  $n$  observations, a  $kd$ -tree is first constructed for the data set. The observation is then queried in the  $kd$ -tree according to the splitting rules until it reaches a terminal node. This process requires a time-complexity of  $O(\log[n])$  on average. Distances of the queried observation to the observations in the terminal node or the node it falls off from are computed to produce  $K$  potential nearest neighbors. These are only potential nearest neighbors because the true nearest neighbors could lie in the nodes that overlap with the minimal hypersphere centered at the queried observation (e.g., the circle centered at the larger, gray square Figure S1(b) overlaps with 4 terminal nodes of the  $kd$ -tree in Figure S1(a)), containing the  $K$  potential nearest neighbors. The exact nearest neighbors can be updated after examining all such overlapping nodes. In general, the expected number of distances that must be computed was shown to be approximately proportional to  $2^k$ . Note that each point cloud is a 3D dataset (i.e.  $k = 3$ ), and so is especially suitable for neighbor searches based on  $kd$ -trees. Thus, the neighbor search algorithm based on  $kd$ -trees has an extremely fast average time-complexity of  $O(\log[n])$  for a fixed dimension.



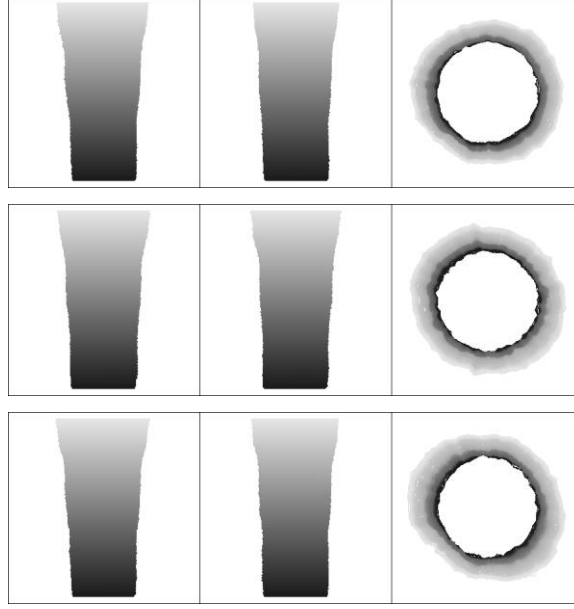
**Figure S1.** Illustration of a  $kd$ -tree for a 2D dataset with 8 observations and a minimal terminal node's size of 1: (a) the constructed  $kd$ -tree, and (b) space partitions corresponding the  $kd$ -tree in Panel (a)

## 8.2 Additional Visualization Plots for the Cylindrical Example in Section 5

This section provides the other two visualization plots, in addition to the visualization plot in Figure 7, for the cylindrical point clouds in Section 5 before removing the rotational registration variability. For the first discovered manifold coordinate, Figure S2 plots three views (front, side, and top) of rendered point clouds corresponding to three points along a parallel line with the first manifold axis. Likewise for the third discovered manifold coordinate, Figure S3 shows the three views of rendered point clouds corresponding to three points along a parallel line with the third manifold axis. It appears that the patterns in Figures S2 and S3 are combinations of the two patterns in the registered data that we see in Figures 9 and 10.



**Figure S2.** Each row shows the front (left column), side (middle column) and top (right column) views of three point clouds selected along the visualization path for the first manifold coordinate. Moving from the top to the bottom row corresponds to increasing the value of the first coordinate, and the grayscale and magnification conventions are the same as in Figure 7.



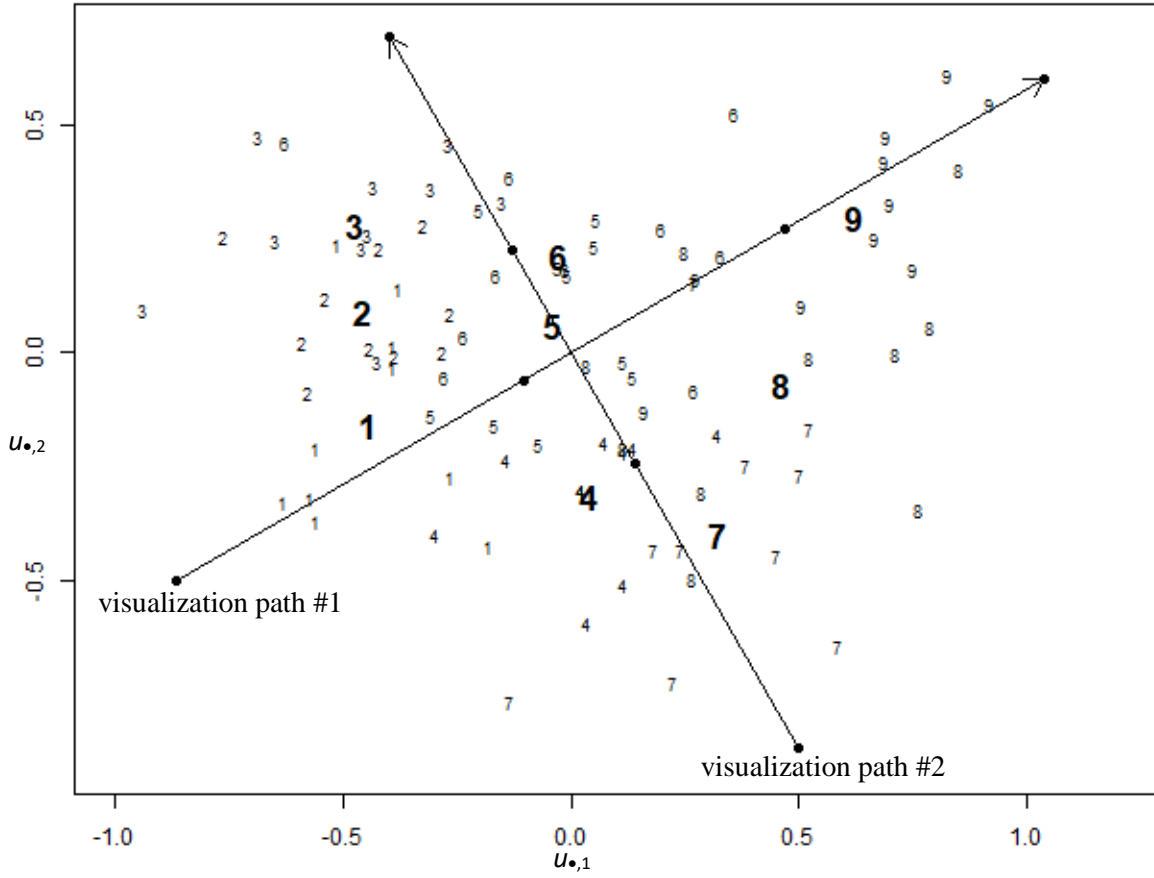
**Figure S3.** Each row shows the front (left column), side (middle column) and top (right column) views of three point clouds selected along the visualization path for the third manifold coordinate. Moving from the top to the bottom row corresponds to increasing the value of the third coordinate, and the grayscale and magnification conventions are the same as in Figure 7.

### 8.3 Revisiting the Cylinder Data Example but Using the Nominal $\mathcal{M}$

This section demonstrates the method in Section 6 with the same set of  $N = 90$  cylindrical point clouds in Section 5 (after rotationally registering them as described in Section 5), but making use of  $\mathcal{M}$ . In Step 1, we define  $n = 420 \times 528 = 221,760$  grid points on the nominal cylinder surface. Then, we structure all the  $N = 90$  cylindrical point clouds based on these grid points, as discussed in Section 6 and obtain  $\mathbf{R}^g = [\hat{\mathbf{r}}_1^g, \hat{\mathbf{r}}_2^g, \dots, \hat{\mathbf{r}}_N^g]^T \in \mathbb{R}^{N \times n}$ , the structured point clouds. There are two parameters that must be selected:  $K_1$  and  $K_2$ . In the following, we illustrate the results with  $K_1 = 4$  (corresponding to a nominal surface mesh with rectangular patches) and  $K_2 = 5$  (which was the value of  $K$  we used in Section 5). After structuring, the Euclidean distances between the structured point clouds  $\mathbf{R}^g$  are used as the point cloud dissimilarities.

In Step 2, we again apply cMDS to these point cloud dissimilarities to compute the manifold coordinates  $\hat{\mathbf{u}}_i$  ( $i = 1, 2, \dots, 90$ ) of the 90 cylinders. The plot of the 20 largest eigenvalues from cMDS in this case is very similar to the right panel of Figure 6 (and therefore is not shown here to save space). This again suggests a dimension of two for  $\hat{\mathbf{u}}$ , the same as in the analysis of the

registered point clouds in Section 5. The 2D MDS coordinates  $\{\hat{\mathbf{u}}_i: i = 1, 2, \dots, N\}$  of the 90 cylinders are plotted in Figure S4, which looks similar to Figure 8 but with the sign of  $u_{\bullet,2}$  reversed. Therefore, we again can observe the same findings described in Section 5. Namely, the learned manifold coordinates agree with the ground truth, in the sense that  $u_{\bullet,1}$  and  $u_{\bullet,2}$  correspond roughly to cutting depth and cutting speed, respectively. As before, the analysis does not use any knowledge of the two known variation sources (depth of cut and speed) in this example.



**Figure S4.** Learned manifold coordinates from applying cMDS to the pairwise dissimilarities of the structured, registered cylindrical point clouds in Section 5 when utilizing the nominal cylinder surface  $\mathcal{M}$ .

In Step 3 we visualize the two identified variation sources above as follows. First, we use PCA to reduce the dimension of the  $N = 90$  structured point clouds  $\mathbf{R}^g$  to  $p = 10$  principal components, which account for about 83% of the total variation in the 90 structured point clouds. Then, we use

a neural network to fit the model in (5) using  $\hat{\mathbf{u}}_i$  as the input and the reduced-dimension  $\tilde{\mathbf{R}}^g$  of  $\mathbf{R}^g$  as the output. The structure of the neural network is 2/30/10 nodes in its three input/hidden/output layers, with sigmoidal activation functions. We use leave-one-out cross-validation to determine the optimal weight decay regularization parameter for the network, which is 0.03 in this example.

Two visualization paths are selected based on the same discussion for the example in Section 5. Traversing along visualization path #1 in Figure S4 corresponds to the circular-to-ellipsoidal pattern in Figure 13 and the animation in file “pattern 1.gif”. Traversing along visualization path #2 in Figure S4 corresponds to the smaller-to-larger pattern in Figure 14 and the animation in file “pattern 2.gif”. Each of these two animations consists of front (left column), side (middle column) and top (right column) views of reconstructed point clouds using the manifold coordinates traversing along the corresponding visualization paths in Figure S4, and the grayscale and magnification conventions are the same as in Figure 7. Note that the directions of both visualization path arrows correspond to decreasing turning speed; however, the cutting depth is increasing/decreasing in the directions of path #1/path #2 in Figure S4.