Efficient, General Point Cloud Registration With Kernel Feature Maps

Hanchen Xiong, Sandor Szedmak, Justus Piater
Institute of Computer Science, University of Innsbruck
Technikerstr.21a A-6020, Innsbruck, Austria
Email: {hanchen.xiong, sandor.szedmak, justus.piater}@uibk.ac.at

***Abstract*—This paper proposes a novel and efficient point cloud registration algorithm based on the kernel-induced feature map. Point clouds are mapped to a high-dimensional (Hilbert) feature space, where they are modeled with Gaussian distributions. A rigid transformation is first computed in feature space by elegantly computing and aligning a small number of eigenvectors with kernel PCA (KPCA) and is then projected back to 3D space by minimizing a consistency error. $SE(3)$ on-manifold optimization is employed to search for the optimal rotation and translation. This is very efficient; once the object-specific eigenvectors have been computed, registration is performed in linear time. Because of the generality of KPCA and $SE(N)$ on-manifold method, the proposed algorithm can be easily extended to registration in any number of dimensions (although we only focus on 3D case). The experimental results show that the proposed algorithm is comparably accurate but much faster than state-of-the-art methods in various challenging registration tasks.**

***Keywords*-kernel method; point cloud registration; SE(3) on-manifold optimization**

I. INTRODUCTION

3D free-form shape registration is an important problem in many fields and has sparked a large volume of related research literature. During the past two decades, 3D point clouds have become an increasingly more important and popular data structure to represent 3D shapes. Especially in contemporary robotics, 3D point cloud registration is an essential component of autonomous systems to assist in the perception of 3D objects and environments.

Until now, most existing 3D point cloud registration algorithms decompose the registration problem into two parts, *correspondence assignment* and *alignment*, because they argue that computing either of two steps will facilitate the other. A popular method is Iterative Closest Point (ICP) [1], which undoubtedly has been most widely used due to its simplicity in implementation. First, a pseudo correspondence is established by finding the nearest neighbor of each point. Then, the optimal rotation and translation are computed so as to minimize the average of Euclidean distances between all pairs of corresponding points. These two steps are iterated until converge. Obviously, the closest-distance criterion for correspondence is too weak, and therefore ICP can easily fail in practice when the displacement between two point clouds or outlier rate is relatively large. To enhance the accuracy of the correspondence, many improved versions of ICP were proposed by incorporating color, normal vectors, curvature, or strategically ignoring some unlikely correspondences [2]. However, despite various improvements, the *hard* assignment strategy employed by ICPs causes problems that require manual assistance in practical applications. To overcome this limitation, SoftAssign [3] and EM-ICP [4] were proposed by establishing one-to-many *soft* correspondences. Both methods assume that one point may correspond to all points in the other cloud with different likelihoods by constructing a correspondence matrix. To iteratively update this matrix, deterministic annealing is used in SoftAssign while an Expectation-Maximization (EM)-style method is employed in EM-ICP. Meanwhile, recently a Gaussian-mixture (GM) method [5] was developed to avoid iteratively computing the correspondences and alignment. GM probabilistically and globally models 3D point clouds as Gaussian mixtures in $\mathbb{R}^3$, and the optimal alignment between point clouds is computed by minimizing the discrepancy (L2 distance [5]) between their corresponding distributions.

For the task of aligning 3D point clouds $\mathbf{M}_1 = \{\mathbf{x}_i^{(1)}\}_{i=1}^{l_1}$ with $\mathbf{M}_2 = \{\mathbf{x}_i^{(2)}\}_{i=1}^{l_2}$, all methods described above can be interpreted as optimizing a common objective function:

$$\{\mathbf{R}^*, \mathbf{b}^*\} = \arg\min_{\mathbf{R},\mathbf{b}} \sum_{i=1}^{l_1}\sum_{j=1}^{l_2} \left(\mathbf{R}\mathbf{x}_i^{(1)} + \mathbf{b} - \mathbf{x}_j^{(2)}\right)^2 w_{i,j} \quad (1)$$

where $w_{i,j}$ denotes the correspondence between every pair of $\mathbf{x}_i^{(1)}$ and $\mathbf{x}_j^{(2)}$. In ICP $w_{i,j} \in \{0,1\}$, and (1) is solved by iteratively updating $w_{i,j}$ in a winner-take-all manner under a shortest-distance criterion and solving a least-squares problem with respect to $\mathbf{R}$ and $\mathbf{b}$. In EM-ICP, $w_{i,j}$ is interpreted as the probability of the correspondence, so a one-way constraint ($w_{i,j} \in [0,1]$, $\sum_j^{l_2} w_{i,j} = 1$) is implicitly imposed. In SoftAssign, a stricter two-way constraint ($w_{i,j} \in [0,1]$, $\sum_j^{l_2} w_{i,j} = 1$, $\sum_i^{l_1} w_{i,j} = 1$) is introduced to enforce globally consistent point correspondences. Although GM does not model the correspondences explicitly, they can likewise be understood as an instance of (1) with Euclidean distance replaced by Mahalanobis distance, and an uniform prior of $w_{i,j} = \frac{1}{l_1 l_2}$ for each pair of i, j. In conclusion, so far 3D point cloud registration can be achieved either by explicitly modeling correspondences and laboriously updating them (EM-ICP and SoftAssign), or

by making some fragile correspondence assumptions to simplify the optimization procedure (ICP and GM). In addition, all these methods share the same computational complexity of $O(n^2)$ [1], which will be a heavy computational burden if the number of points n is relatively large. Therefore, a registration solution that can both express realistic priors over point correspondence matches and can be computed in a simpler (possibly non-iterative) and cheaper (possibly non-quadratic time) way is highly desirable. The method proposed in this paper fulfills both demands. Instead of doing point-wise correspondence search and computing in 3D space, our method first maps all points to a higher-dimensional (reproducing kernel Hilbert) feature space using kernel methods. The optimal transformation in feature space is then found by aligning Gaussians that approximate the two mapped point clouds. To project back to the 3D space, an objective function is constructed based on the fact that the transformed mapped points should be consistent with mapped transformed points. Finally, an $SE(3)$ on-manifold optimization scheme is exploited to provide an elegant and efficient gradient-type algorithm for registration.

Compared to previous registration methods, the strength of our method can be summarized in four points: (1) Although our algorithm was not developed on the basis of point correspondences, the form of its objective function (section III-A) suggests that correspondences are implicitly derived and to large degree it is consistent with the correct matches; (2) The experimental results (section IV) show that our method can work accurately and robustly in various challenging cases (large motion, outlier points); (3) Our method is much more efficient than other state-of-the-art methods, actually the computation complexity is $O(n \log n)$; (4) By using Kernel PCA and $SE(3)$ on-manifold optimization, the algorithm can be used as general point cloud registration framework with high flexibility and extensibility to any dimension.

II. RIGID TRANSFORMATION IN HILBERT SPACE

Intuitively, a straightforward way to align point clouds without point-wise correspondences is to first probabilistically fit each point cloud to a single Gaussian distribution in $\mathbb{R}^3$ and then align their means (translation) and covariances (rotation). However, the modeling ability of one single Gaussian in 3D space is too limited to capture the 3D point distribution of real-world objects, i.e. the mean and covariance in $\mathbb{R}^3$ are very sensitive to outliers. Inspired by kernel methods developed for set-format data [6], a 3D point clouds can be implicitly mapped to a much higher-dimensional Hilbert feature space, where a single Gaussian can fit well (Fig. 1) and hence yields higher tolerance to 3D disturbance in the original point cloud (e.g. outliers or non-rigid transformation). In addition, by applying kernel PCA, the eigenvectors of covariances can be efficiently computed and aligned without explicit computation in feature space.

[1] the complexity of ICP is $O(n \log n)$ if K-d trees are used for searching for the nearest neighbour

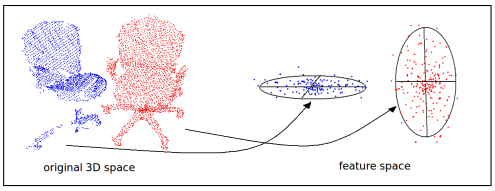


Figure 1. Mapping point clouds from 3D space to an infinite-dimensional Hilbert space, where a single Gaussian is sufficient to model distributions of complex shape.

A. Probabilistic modeling in Hilbert space

Inspired by kernel methods that have been widely used in machine learning, in order to map all points in a point cloud $\mathbf{M} = \{\mathbf{x}_i\}_{i=1}^{l}$ to a higher, possibly infinite-dimensional feature space, we can define a kernel function on 3D points $K(\mathbf{x}_i, \mathbf{x}_j)$. Then, a feature map can be implicitly induced by satisfying

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle \tag{2}$$

where ϕ is the corresponding feature map: $\mathbb{R}^3 \to \mathcal{H}$, and $\mathcal{H}$ is referred to as the reproducing Hilbert feature space. Since the structure of $\mathbf{M}$ can be far too complicated in $\mathbb{R}^3$, to ensure that one single Gaussian is capable of modeling the distribution of $\{\phi(\mathbf{x}_i)\}_{i=1}^{l}$ in $\mathcal{H}$, in this paper we select the kernel function as the radial basis function (RBF)

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp \frac{-\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2} \tag{3}$$

because the induced feature map is a scaled Gaussian probability density function (PDF),

$$\phi(\mathbf{x}_i) = f(\xi|\mathbf{x}_i) = \exp \frac{-\|\xi - \mathbf{x}_i\|^2}{2\sigma^2}, \tag{4}$$

i.e., $\phi(\cdot)$ corresponds to an infinite-dimensional feature map.

With all points mapped into feature space, a Gaussian (mean and covariance) in $\mathcal{H}$ can be easily fitted by using maximum likelihood estimation (MLE):

$$\boldsymbol{\mu}_{\mathcal{H}} = \frac{1}{l} \sum_{i=1}^{l} \phi(\mathbf{x}_i) = \frac{1}{l} \phi(\mathbf{M})^\top \mathbf{1}_l \tag{5}$$

$$\boldsymbol{\Sigma}_{\mathcal{H}} = \frac{1}{l} \sum_{i=1}^{l} \left(\phi(\mathbf{x}_i) - \boldsymbol{\mu}_{\mathcal{H}}\right) \left(\phi(\mathbf{x}_i) - \boldsymbol{\mu}_{\mathcal{H}}\right)^\top \tag{6}$$

where $\phi(\mathbf{M})^\top = [\phi(\mathbf{x}_1), \phi(\mathbf{x}_2), \cdots, \phi(\mathbf{x}_l)]$ and $\mathbf{1}_l$ is an l-dimensional vector with all elements equal to 1.

B. Kernel PCA

To achieve the alignment between two covariances, their eigenvectors should be computed first. However, this computation is non-trivial in feature space. Kernel principal component analysis (KPCA) [7] is a technique developed

to compute eigenvectors in feature space without explicit computation in $\mathcal{H}$. Here we briefly review the procedure of KPCA with its application to 3D point clouds.

Assuming all points are already centered in feature space, the covariance $\mathbf{\Sigma}_{\mathcal{H}}$ of the Gaussian can be estimated as

$$\mathbf{\Sigma}_{\mathcal{H}} = \frac{1}{l}\sum_{i=1}^{l}\phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^\top \tag{7}$$

which is a symmetric bilinear form on $\mathcal{H}$. Analogous to the symmetric covariance matrices in the finite-dimensional case, its nonzero eigenvalue λ_k and corresponding eigenvector $\mathbf{u}_k$ should satisfy

$$\lambda_k \mathbf{u}_k = \mathbf{\Sigma}_{\mathcal{H}}\mathbf{u}_k. \tag{8}$$

By substituting (7) into (8), we have

$$\mathbf{u}_k = \frac{1}{\lambda_k}\mathbf{\Sigma}_{\mathcal{H}}\mathbf{u}_k = \sum_{i=1}^{l}\alpha_i^k\phi(\mathbf{x}_i) \tag{9}$$

where $\alpha_i^k = \frac{\phi(\mathbf{x}_i)^\top\mathbf{u}_k}{\lambda_k l}$. Therefore, all eigenvectors $\mathbf{u}_k$ with $\lambda_k \neq 0$ must lie in the span of $\phi(\mathbf{x}_1), \phi(\mathbf{x}_2), \ldots, \phi(\mathbf{x}_l)$, and (9) is referred to as the dual form of $\mathbf{u}_k$. By left-multiplying $\sum_{j=1}^{l}\phi(\mathbf{x}_j)^\top$ on both sides of equation (8), we have

$$\begin{aligned} & \sum_{j=1}^{l}\phi(\mathbf{x}_j)^\top\lambda_k\mathbf{u}_k &=& \sum_{j=1}^{l}\phi(\mathbf{x}_j)^\top\mathbf{\Sigma}_{\mathcal{H}}\mathbf{u}_k \\ \Leftrightarrow\quad & \lambda_k\sum_{i,j=1}^{l}\alpha_i^k K(\mathbf{x}_i,\mathbf{x}_j) &=& \frac{1}{l}\sum_{i,j=1}^{l}\alpha_i^k K(\mathbf{x}_i,\mathbf{x}_j)^2 \\ \Leftrightarrow\quad & l\lambda_k\boldsymbol{\alpha}^k &=& \mathbf{K}\boldsymbol{\alpha}^k \end{aligned} \tag{10}$$

where $\mathbf{K}$ is an $l \times l$ kernel matrix with $\mathbf{K}_{ij} = K(x_i, x_j)$, $\boldsymbol{\alpha}^k = (\alpha_1^k, \alpha_2^k, \ldots, \alpha_l^k)^\top$. It can be seen that $\{\boldsymbol{\alpha}^k, \eta^k = l\lambda_k\}$ is actually an eigenvalue-eigenvector pair of matrix $\mathbf{K}$. Therefore, by using dual forms of eigenvectors, the eigenvector decomposition of $\mathbf{\Sigma}_{\mathcal{H}}$ can be transformed to the decomposition of the finite matrix $\mathbf{K}$. In addition, because all $\mathbf{u}_k$ should be unit vectors:

$$1 = \mathbf{u}_k^\top\mathbf{u}_k = \langle\boldsymbol{\alpha}^k, \mathbf{K}\boldsymbol{\alpha}^k\rangle = \eta^k\boldsymbol{\alpha}^{k\top}\boldsymbol{\alpha}^k \tag{11}$$

the $\boldsymbol{\alpha}^k$ should be normalized as:

$$\boldsymbol{\alpha}^k \leftarrow \frac{\boldsymbol{\alpha}^k}{\sqrt{\eta^k}} \tag{12}$$

However, though the point cloud can be easily centered in 3D space, it does not mean it is also centered in feature space. By replacing $\phi(\mathbf{x}_i)$ with $\tilde{\phi}(\mathbf{x}_i) = \phi(\mathbf{x}_i) - \boldsymbol{\mu}$, the corresponding kernel matrix $\tilde{\mathbf{K}}$ is

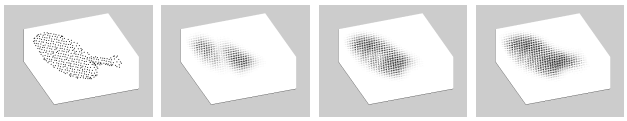
$$\tilde{\mathbf{K}} = \mathbf{K} - \frac{1}{l}\mathbf{E}\mathbf{K} - \frac{1}{l}\mathbf{K}\mathbf{E} + \frac{1}{l^2}\mathbf{E}\mathbf{K}\mathbf{E} \tag{13}$$

where $\mathbf{E}$ denotes an $l \times l$ matrix with all entries equal to 1. After similar eigenvector decomposition (10) and normalization (12) steps, we obtain eigenvectors

$$\tilde{\mathbf{u}}_k = \sum_{i=1}^{l}\tilde{\alpha}_i^k\left(\phi(\mathbf{x}_i) - \boldsymbol{\mu}\right) = \phi(\mathbf{M})^\top\underbrace{\left(\mathbf{I}_l - \frac{1}{l}\mathbf{E}\right)}_{\mathbf{I^E}}\tilde{\boldsymbol{\alpha}}^k \tag{14}$$

where $\mathbf{I}_l$ is an $l \times l$ identity matrix and $\tilde{\boldsymbol{\alpha}}$ is the kth eigenvector of the matrix $\tilde{\mathbf{K}}$.

As analyzed in [6], it is misleading and wasteful to use full covariances, so only a small number of eigenvectors are sufficient to capture the structural property of the covariance $\mathbf{\Sigma}_{\mathcal{H}}$. Fig. 2 displays an example of a table tennis racket point cloud. Its Gaussian distribution in the feature space $\mathcal{H}$ can be well reconstructed using only 3 of its principal components associated with top largest eigenvalues.



(a) (b) (c) (d)

Figure 2. (a) A point cloud of table tennis racket; (b–d) reconstruction using the first 1–3 principal components. For each point in the bounding-box volume, the darkness is proportional to the density of the Gaussian in the feature space $\mathcal{H}$.

C. Alignment of Gaussians

Assume the task is to align a point cloud $\mathbf{M}_1 = \{\mathbf{x}_i^{(1)}\}_{i=1}^{l_1}$ with $\mathbf{M}_2 = \{\mathbf{x}_j^{(2)}\}_{j=1}^{l_2}$, instead of computing the optimal alignment in 3D space directly, we can alternatively first align them in feature space, and then project them back to $\mathbb{R}^3$ (section III). With the modeling procedure above applied on $\mathbf{M}_1$ and $\mathbf{M}_2$, the alignment of two point clouds in feature space corresponds to aligning two Gaussians. In this paper, we assume $\mathbf{D}$ eigenvectors are computed for the covariance of each point cloud: $\tilde{\mathbf{U}}_1 = \left[\tilde{\mathbf{u}}_1^1, \ldots, \tilde{\mathbf{u}}_1^k, \ldots, \tilde{\mathbf{u}}_1^{\mathbf{D}}\right]$, $\tilde{\mathbf{U}}_2 = \left[\tilde{\mathbf{u}}_2^1, \ldots, \tilde{\mathbf{u}}_2^k, \ldots, \tilde{\mathbf{u}}_2^{\mathbf{D}}\right]$. Therefore, the rotation in feature space $\mathbf{R}_{\mathcal{H}}$ is estimated by simultaneously aligning $\mathbf{D}$ pairs of corresponding eigenvectors: $\tilde{\mathbf{U}}_2 = \mathbf{R}_{\mathcal{H}}\tilde{\mathbf{U}}_1$. Because different eigenvectors of each point cloud are orthogonal to each other, based on the computation result in (14), it is easy to derive:

$$\begin{aligned} \mathbf{R}_{\mathcal{H}} &= \tilde{\mathbf{U}}_2\tilde{\mathbf{U}}_1^\top \\ &= \phi(\mathbf{M}_2)^\top\underbrace{\mathbf{I}_2^{\mathbf{E}}\left(\sum_{k=1}^{\mathbf{D}}\tilde{\boldsymbol{\alpha}}_2^k\tilde{\boldsymbol{\alpha}}_1^{k\top}\right)\mathbf{I}_1^{\mathbf{E}}}_{\mathbf{\Theta}_{\boldsymbol{\alpha}}}\phi(\mathbf{M}_1) \end{aligned} \tag{15}$$

Since the rotation (15) can be applied only if $\mathbf{M}_1$ has already been centered in feature space, to fully align two Gaussians, the translation in feature space $\mathbf{b}_{\mathcal{H}}$ obviously should equal the mean of the Gaussian that models $\mathbf{M}_2$ in feature space:

$$\mathbf{b}_{\mathcal{H}} = \boldsymbol{\mu}_{\mathcal{H}}^{(2)} = \frac{1}{l_2}\phi(\mathbf{M}_2)^\top\mathbf{1}_{l_2} \tag{16}$$

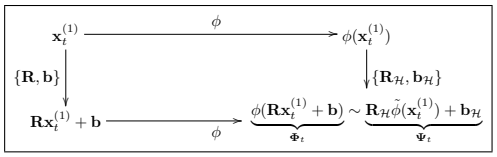


Figure 3. The consistency error is defined as the discrepancy between $\phi(\mathbf{R}x_t + \mathbf{b})$ and $\mathbf{R}_\mathcal{H}\tilde{\phi}(x_t) + \mathbf{b}_\mathcal{H}$.

Now we can align two Gaussians in feature space with $\{\mathbf{R}_\mathcal{H}, \mathbf{b}_\mathcal{H}\}$ computed in (15) and (16). However, due to the infinite-dimensional feature map defined in (4), there still exist two obstacles to be overcome: First, (15) and (16) cannot be computed in an analytic form; secondly, there is no trivial way to map $\{\mathbf{R}_\mathcal{H}, \mathbf{b}_\mathcal{H}\}$ back to 3D space. Fortunately, by designing a consistency error (section III-A), these two issues can surprisingly be solved simultaneously in a very smooth and elegant manner.

III. CARTESIAN POINT CLOUD ALIGNMENT

In this section we will project the rotation and translation in the feature space $\mathcal{H}$ back to 3D space by minimizing a specifically-designed consistency error (Fig. 3). It turns out that the final objective function can be constructed and solved without explicit computation in feature space. In addition, further connections with other registration methods will be exposed by discovering the hidden commonality among their objective functions. To enhance the generality of the proposed algorithm, an $SE(3)$ on-manifold optimization scheme is employed to search for the optimal transformation.

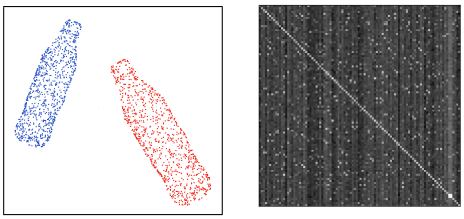
A. Consistency Error

Instead of tediously finding the inverse function $\phi^{-1}(\cdot)$ corresponding to the definition in (4) and applying it to $\{\mathbf{R}_\mathcal{H}, \mathbf{b}_\mathcal{H}\}$ to map them back to 3D space $\{\mathbf{R}, \mathbf{b}\}$, here we define a novel consistency error between $\phi(\mathbf{R}x_t + \mathbf{b})$ and $\mathbf{R}_\mathcal{H}\left(\phi(x_t) - \boldsymbol{\mu}_1\right) + \mathbf{b}_\mathcal{H}$ based on the fact that mapping after transformation should be consistent with transformation after mapping (Fig. 3). Therefore, we can find the optimal rotation and translation in 3D space by minimizing the average consistency error:

$$\{\mathbf{R}^*, \mathbf{b}^*\} = \arg\min_{\mathbf{R},\mathbf{b}} \frac{1}{l_1}\sum_{t=1}^{l_1} \|\mathbf{\Psi}_t - \mathbf{\Phi}_t\|^2 \tag{17}$$

Because $\|\mathbf{\Phi}(\boldsymbol{x})\|^2$ is the integration over the square of a Gaussian, which preserves constant under any translation $\mathbf{b}$ and rotation $\mathbf{R}$, and $\mathbf{\Psi}_t$ is fixed, by substituting (15) and (16) into (17), we have

$$\begin{aligned}\{\mathbf{R}^*, \mathbf{b}^*\} &= \arg\max_{\mathbf{R},\mathbf{b}} \frac{1}{l_1}\sum_{t=1}^{l_1} \mathbf{\Psi}_t^\top \mathbf{\Phi}_t \\ &= \arg\max_{\mathbf{R},\mathbf{b}} \underbrace{\frac{1}{l_1}\sum_{t=1}^{l_1} K(\mathbf{R}\mathbf{x}_t^{(1)} + \mathbf{b}, \mathbf{M}_2)^\top \boldsymbol{\rho}_t}_{\mathbf{O}}\end{aligned} \tag{18}$$



(a) (b)

Figure 4. (a) Two identical point clouds with exactly the same point permutation. (b) Visualization of ρ_{ti} computed for all pairs of points.

$$\boldsymbol{\rho}_t = \mathbf{\Theta}_{\boldsymbol{\alpha}}\left(K(\mathbf{x}_t^{(1)}, \mathbf{M}_1) - \frac{1}{l_1}\mathbf{K}_1\mathbf{1}_{l_1}\right) + \frac{1}{l_2}\mathbf{1}_{l_2} \tag{19}$$

where $K(\mathbf{R}\mathbf{x}_t^{(1)} + \mathbf{b}, \mathbf{M}_2)$ is an l_2-dimensional vector with $K(\mathbf{R}\mathbf{x}_t^{(1)} + \mathbf{b}, \mathbf{M}_2)_i = K(\mathbf{R}\mathbf{x}_t^{(1)} + \mathbf{b}, \mathbf{x}_i^{(2)})$, and $K(\mathbf{x}_t^{(1)}, \mathbf{M}_1)$ is an l_1- dimensional vector with $K(\mathbf{x}_t^{(1)}, \mathbf{M}_1)_j = K(\mathbf{x}_t^{(1)}, \mathbf{x}_j^{(1)})$. It can be seen that by employing the kernel trick (2), we can elegantly avoid computation in the feature space $\mathcal{H}$ in both (18) and (19).

B. Relation to Other Approaches

As analyzed in section I, most existing registration methods can be unified to a general objective function (1) with different correspondence assumptions or iterative update strategies. The objective function (18) can be easily extended as follows:

$$\{\mathbf{R}^*, \mathbf{b}^*\} = \arg\min_{\mathbf{R},\mathbf{b}} \sum_{t=1}^{l_1}\sum_{i=1}^{l_2} -K(\mathbf{R}\mathbf{x}_t^{(1)} + \mathbf{b}, \mathbf{x}_i^{(2)})\rho_{t,i} \tag{20}$$

By considering $-K(\cdot,\cdot)$ as an exponential distance and replacing $\rho_{t,i}$ with $w_{t,i}$, it turns out that surprisingly our method (20) is also a special case of (1), although we arrive there from a completely different starting point. This suggests that $\rho_{t,i}$ somehow implicitly encodes the correspondence likelihood between $\mathbf{x}_t^{(1)}$ and $\mathbf{x}_i^{(2)}$ as well. To verify this argument experimentally, in Fig 4(a) there are two identical point clouds with exactly the same point permutation. We compute ρ_{ti} for all pairs of points in Fig 4(b). It can be seen that Fig 4(b) shows a very evident diagonal pattern with uniformly distributed noise, which is the reflection of our prior knowledge. However, different from most of other approaches, we do not model $w_{t,i}$ explicitly or update them iteratively. Instead, $\rho_{t,i}$ is derived from eigenvector alignment in feature space and only need to be computed once.

There is another way our method is related to Gaussian mixtures. By relaxing the non-negative coefficient constraint in the definition of Gaussian mixtures, each eigenvector in (14) can be considered as a pseudo Gaussian mixture with $\phi(\cdot)$ defined as in (4). In this way, instead of aligning two Gaussian mixtures in 3D space, what our method is actually doing is to simultaneously align $\mathbf{D}$ pairs of Gaussian

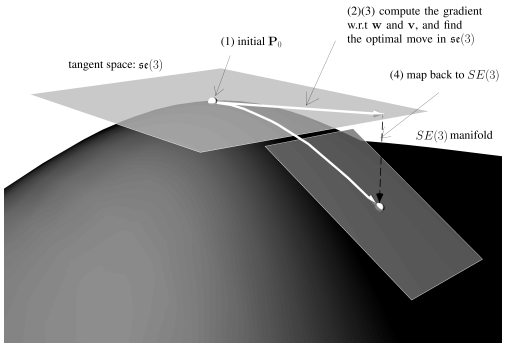


Figure 5. The $SE(3)$ manifold and its optimization scheme: (1) start from a rotation matrix $\mathbf{P}_0$; (2) use equation (26) as the local parametrization of the manifold at point $\mathbf{P}_0$, and compute the gradient with respect to $\{\mathbf{w}, \mathbf{v}\}$; (3) compute the best move in $\mathfrak{se}(3)$ by mapping the update of $\{\mathbf{w}, \mathbf{v}\}$; (4) map back to $SE(3)$: $\mathbf{P}_1 \leftarrow \exp(\Lambda)\mathbf{P}_0$; (5) repeat step (2)(3)(4) until convergence

mixtures in feature space, and then implicitly maps back to original 3D space.

C. SE(3) on-manifold optimization

When solving the optimization problem (18), the orthogonality constraint of the rotation matrix $\mathbf{R}$ must be taken into account: $\mathbf{R}^\top\mathbf{R} = \mathbf{I}$. This constraint is a common obstacle in various rotation-related optimization problems, which drove many researchers to alternatives such as unit quaternions [5] or dual quaternion number [8]. However, although all these methods can work satisfactorily for 3D rotation, they are difficult to be extend to higher dimensions. Although we only focus 3D point cloud registration here, to make our algorithm more general, here we employ an $SE(3)$ on-manifold optimization scheme [9], which can be easily adapted to rotation matrices in any dimension. Another virtue of $SE(3)$ on-manifold optimization is that combined with gradient computing, an elegant optimizer can be developed based on its associated Lie algebra to circumvent the orthogonality constraint.

Each rotation and translation in 3D space $\{\mathbf{R}, \mathbf{b}\}$ can be jointly treated as a Euclidean transformation $\mathbf{P}$ in $\mathbb{R}^3$ by using homogeneous coordinates. From now on, $\mathbf{x}$ is used to denote homogeneous coordinate, and $\overline{\mathbf{x}}$ for the original one:

$$\mathbf{x}^\top = [\overline{\mathbf{x}}^\top, 1] \tag{21}$$

and correspondingly,

$$\mathbf{P} = \begin{bmatrix} \mathbf{R} & \mathbf{b} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{22}$$

One specific $\mathbf{P}$ is actually an element of Lie group $SE(3)$ (Special Euclidean Group), which is a smooth manifold embedded in $\mathbb{R}^3$. Intuitively, the $SE(3)$ manifold can be considered as a topological space wherein all points are 4×4 Euclidean transformation matrices, and at each point, there exists a tangent space Λ, which happens to be its associated Lie algebra $\mathfrak{se}(3)$. The mathematical connection between $SE(3)$ and $\mathfrak{se}(3)$ is

$$\mathfrak{se}(3) \to SE(3) : \mathbf{P} = \exp(\Lambda) \tag{23}$$

where $\exp(\cdot)$ denotes the exponential map. The tangent space $\mathfrak{se}(3)$ can be considered as a linearization of the $SE(3)$ manifold within the infinitesimally small vicinity of certain point $\mathbf{P}_0$, so inversely, the exponential map works as a 'de-linearization'. All concepts described above are illustrated in Fig. 5. The Lie algebra $\mathfrak{se}(3)$ is a collection of matrices of the form

$$\Lambda = \begin{bmatrix} J(\mathbf{w}) & \mathbf{v} \\ \mathbf{0}_{1\times 3} & 0 \end{bmatrix} \tag{24}$$

where $J(\mathbf{w})$ is an skew-symmetric matrix, which can be constructed from a 3D vector $\mathbf{w}$ with a skew operator $J(\cdot)$:

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} \to J(\mathbf{w}) = \begin{bmatrix} 0 & -w_3 & w_2 \\ w_3 & 0 & -w_1 \\ -w_2 & w_1 & 0 \end{bmatrix} \tag{25}$$

and $\mathbf{v}$ is an usual 3D vector. Therefore, Therefore, when $\exp(\Lambda) \to \mathbf{I}_3$ (e.g. computing gradient), we can establish a straightforward map from $\mathbb{R}^6$ to the local neighboring region of $\mathbf{P}_0$ on manifold as

$$\mathbf{P} = \exp\left(\begin{bmatrix} J(\mathbf{w}) & \mathbf{v} \\ \mathbf{0}_{1\times 3} & 0 \end{bmatrix}\right) \cdot \mathbf{P}_0 \tag{26}$$

Last but not least, combination with a gradient-type method yields an final optimization procedure for $SE(3)$ parameters. By using (26), we can see that when computing the gradient with respect to $\mathbf{w}$ and $\mathbf{v}$, the orthogonality constraint will be avoided, and therefore, the constrained optimization problem in $SE(3)$ (18) is naturally and smoothly transformed to a much simpler, unconstrained problem in $\mathbb{R}^6$. Meanwhile, different from conventional gradient methods, instead of computing gradient and updating within the same space, in $SE(3)$ on-manifold optimization, after every update of $\{\mathbf{w}, \mathbf{v}\}$, it need to be mapped back to $SE(3)$, and subsequently the gradient is computed with the local parametrization of the corresponding neighboring region. The whole procedure of $SE(3)$ on-manifold optimization scheme is illustrated in Fig. 5.

D. Reduction of Computational Complexity

If we reexamine the objective function $\mathbf{O}$ (18), an interesting property can be leveraged to significantly reduce the computation complexity:

$$\begin{aligned} & \langle \mathbf{\Phi}_t, \mathbf{\Psi}_t \rangle \\ = \; & \phi(\overline{\mathbf{P}\mathbf{x}_t^{(1)}})^\top \left(\sum_{k=1}^{\mathbf{D}} \tilde{\mathbf{u}}_2^k \tilde{\mathbf{u}}_1^{k\top} \left(\overline{\phi(\mathbf{x}_t^{(1)}) - \boldsymbol{\mu}_1} \right) + \boldsymbol{\mu}_2 \right) \\ = \; & \sum_{k=1}^{\mathbf{D}} \langle \tilde{\boldsymbol{u}}_2^k, \phi(\overline{\mathbf{P}\mathbf{x}_t^{(1)}}) \rangle \langle \tilde{\boldsymbol{u}}_1^k, \overline{\phi(\mathbf{x}_t^{(1)}) - \boldsymbol{\mu}_1} \rangle + \langle \boldsymbol{\mu}_2, \phi(\overline{\mathbf{P}\mathbf{x}_t^{(1)}}) \rangle \\ = \; & \sum_{k=1}^{\mathbf{D}} \langle \tilde{\boldsymbol{u}}_2^k, \phi(\overline{\mathbf{P}\mathbf{x}_t^{(1)}}) \rangle \langle \tilde{\boldsymbol{u}}_2^k, \mathbf{R}_{\mathcal{H}} \overline{\phi(\mathbf{x}_t^{(1)}) - \boldsymbol{\mu}_1} \rangle + \langle \boldsymbol{\mu}_2, \phi(\overline{\mathbf{P}\mathbf{x}_t^{(1)}}) \rangle \end{aligned} \tag{27}$$

where we can see that $\overline{\phi(\mathbf{P}\mathbf{x}_t^{(1)})}$ and $\overline{\mathbf{R}_{\mathcal{H}}\phi(\mathbf{x}_t^{(1)}) - \boldsymbol{\mu}_1}$ are projected onto $\mathbf{D}$ eigenvectors $\left\{\tilde{\boldsymbol{u}}_2^k\right\}_{k=1}^{\mathbf{D}}$ respectively, and an additional projection of $\phi(\mathbf{P}\mathbf{x}_t^{(1)})$ onto $\boldsymbol{\mu}_2$. Therefore, the computation of the objective function is actually done in a space spanned by $\mathbf{D}$ eigenvectors $\left\{\tilde{\boldsymbol{u}}_2^k\right\}_{k=1}^{\mathbf{D}}$ and one $\boldsymbol{\mu}_2$, which is a subspace of $\mathcal{H}$. We denote this subspace by $\mathcal{S}$. The feature map of each point $\phi(\mathbf{x}_i)$ can be projected onto $\mathcal{S}$ in the following way:

$$\mathcal{S}\left(\phi(\mathbf{x}_i)\right) = \sum_{k}^{\mathbf{D+1}} \beta_{i,k}\mathbf{r}_k \tag{28}$$

where $\mathbf{r}_k$ are referred to as $\mathbf{D}+1$ reference vectors in $\mathcal{S}$, and $\beta_{i,k}$ are the corresponding coefficients used to express $\mathcal{S}\left(\phi(\mathbf{x}_i)\right)$. In other words, in $\mathcal{S}$, only $\mathbf{D}+1$ reference vectors, of which $\mathbf{D}$ should be linearly independent, can represent any $\mathcal{S}(\phi(\mathbf{x}_i))$. Therefore, to ensure that $\boldsymbol{\Phi}_t$ and $\boldsymbol{\Psi}_t$ are consistent with each other for all points $\mathbf{x}_t^{(1)}$ in $\mathbf{M}_1$, we only need to align $\mathbf{D}+1$ predefined reference vectors. In practice, we can randomly select $\mathbf{D}+1$ $\mathcal{S}(\phi(\mathbf{x}_i))$ because they are very likely to be linear independent in $\mathcal{S}$. Thus, the objective function can be simplified to

$$\{\mathbf{R}^*, \mathbf{b}^*\} = \arg\max_{\mathbf{R},\mathbf{b}} \frac{1}{\mathbf{D}+1}\sum_{t=1}^{\mathbf{D+1}} K(\mathbf{R}\mathbf{x}_{\mathbf{S}_t}^{(1)} + \mathbf{b}, M_2)^\top \boldsymbol{\rho}_t \tag{29}$$

where $\mathbf{S}$ denotes the randomly selected subset of $\mathbf{M}_1$. To practically apply $SE(3)$ on-manifold optimization on the objective function $\mathbf{O}$ (29), we compute the derivatives of (29) w.r.t. $\mathbf{w}$ and $\mathbf{v}$ as follows:

$$\frac{d\mathbf{O}}{d\left[\mathbf{w}^\top, \mathbf{v}^\top\right]^\top} = \frac{1}{\mathbf{D}+1}\sum_{t=1}^{\mathbf{D+1}}\left(\frac{d\mathbf{O}}{d\mathbf{P}\mathbf{x}_{\mathbf{S}_t}^{(1)}}\frac{d\mathbf{P}\mathbf{x}_{\mathbf{S}_t}^{(1)}}{d\left[\mathbf{w}^\top, \mathbf{v}^\top\right]^\top}\right) \tag{30}$$

where

$$\frac{d\mathbf{O}}{d\mathbf{P}\mathbf{x}_t^{(1)}} = \sum_{j=1}^{l_2}\rho_{\mathbf{S}_t,j}K\left(\mathbf{P}\mathbf{x}_{\mathbf{S}_t}^{(1)}, \mathbf{x}_j^{(2)}\right)\frac{1}{\sigma^2}\left(\mathbf{x}_j^{(2)} - \mathbf{P}\mathbf{x}_{\mathbf{S}_t}^{(1)}\right)^\top \tag{31}$$

$$\frac{\partial\mathbf{P}\mathbf{x}_{\mathbf{S}_t}^{(1)}}{\partial\mathbf{w}} = \frac{\partial\exp(\Lambda)\mathbf{P}_0\mathbf{x}_{\mathbf{S}_t}^{(1)}}{\partial\mathbf{w}} = \left[J(\overline{\mathbf{P}_0\mathbf{x}_{\mathbf{S}_t}^{(1)}}), \mathbf{0}_{3\times1}\right]^\top \tag{32}$$

$$\frac{\partial\mathbf{P}\mathbf{x}_{\mathbf{S}_t}^{(1)}}{\partial\mathbf{v}} = \frac{\partial\exp(\Lambda)\mathbf{P}_0\mathbf{x}_t^{(1)}}{\partial\mathbf{v}} = \left[\mathbf{I}_3, \mathbf{0}_{3\times1}\right]^\top \tag{33}$$

IV. EXPERIMENTS

A. Implementation details

Since the computed eigenvectors (14) are of no direction, there could be $2^{\mathbf{D}}$ possible alignments for $\mathbf{D}$ eigenvectors in feature space. Fortunately, according to experiment, we found that $\mathbf{D} = 3$ is actually enough to make sufficiently good alignment. Therefore, one has to compute all 8 possible alignments in feature space and project them back to $\mathbb{R}^3$, then the final optimal one is picked by checking the accumulated distances between every pair of corresponding points in two clouds, and we use shortest distance as the correspondence here. An outline of the proposed algorithm is given in Algorithm 1. In practice, to speed up the convergence of the algorithm, some sophisticated stepsize tricks can also be added. In addition, we also find that in Algorithm 1 computing eigenvectors (line 2) is the most time-consuming part, so in our implementation, fast-PCA [10] is employed to accelerate the computation.

Algorithm 1 3D Point Cloud Registration

Input: $\mathbf{M}_1 = \{\mathbf{x}_i^{(1)}\}_{i=1}^{l_1}$ and $\mathbf{M}_2 = \{\mathbf{x}_j^{(2)}\}_{j=1}^{l_2}$, $\mathbf{x} \in \mathbb{R}^3$
Output: the optimal motion estimation $\mathbf{P}^*$ which can align $\mathbf{M}_1$ with $\mathbf{M}_2$

1: construct two matrices $\tilde{\mathbf{K}}_1$ and $\tilde{\mathbf{K}}_2$ (13)
2: compute eigenvalue-eigenvector pairs for $\tilde{\mathbf{K}}_1$ and $\tilde{\mathbf{K}}_2$: $\{\boldsymbol{\alpha}_{m,k}, \eta_{m,k}\}$ $m = 1, 2$
3: normalize eigenvectors (12)
4: select $\mathbf{D} = 3$ eigenvectors with largest eigenvalues for both $\mathbf{M}_1$ and $\mathbf{M}_2$
5: randomly select a subset of $\mathbf{N} \geq \mathbf{D}+1$ size from $\mathbf{M}_1$
6: set initial $\mathbf{P}_0$ randomly
7: compute $\boldsymbol{\Theta}_{\boldsymbol{\alpha}}$ (15) with the subset
8: **while** 1 **do**
9: compute the gradient $\nabla_{\mathbf{w}}$ and $\nabla_{\mathbf{w}}$ with current $\mathbf{P}_n$ (30–33)
10: **if** both $\nabla_{\mathbf{w}}$ and $\nabla_{\mathbf{v}}$ are small enough **then**
11: **return** $\mathbf{P}_n$
12: **end if**
13: map the update of $\mathbf{w}$ and $\mathbf{v}$ back to $SE(3)$ (26)
14: set $n \leftarrow n+1$
15: **end while**
16: repeat line 7–15 $2^{\mathbf{D}}$ times with different sign combinations of eigenvectors, and select the final optimal $\mathbf{P}^*$ which yields the minimal accumulated distances between every pair of closest points in $\mathbf{P}_n\mathbf{M}_1$ and $\mathbf{M}_2$

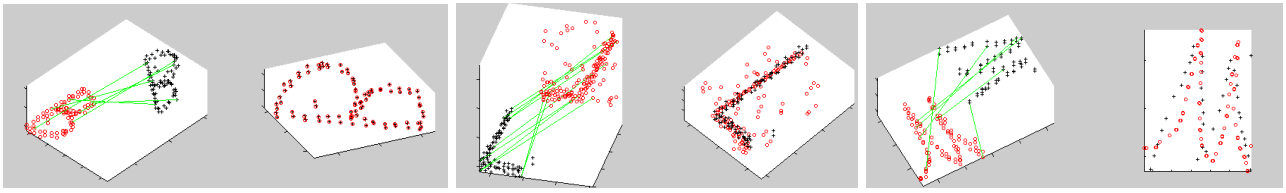
B. Qualitative Evaluation

For the sake of visualization, we first test our algorithm on some toy point clouds to see how it work qualitatively. In Figure 6, some test examples on handwritten letters are displayed. It can be seen that in rather challenging circumstances, i.e. (1) the motion between two point clouds is arbitrarily large (Figure 6(a)), (2) a large portion of outliers are added (Figure 6(b)), (3) nonrigid transformation is applied (Figure 6(c)), the proposed algorithm can still discover roughly correct corresponding points [2] between two point clouds (green lines in Figure 6) and make qualitatively acceptable alignment.

C. Quantitative Evaluation

To obtain a more precise and convincing evaluation of the the proposed algorithm, KIT database [11] is used for

[2] the correspondence for point $\mathbf{x}_t^{(1)}$ is determined by finding the index $j^* = \arg\max_{j\in[1,l_2]}\rho_{tj}$



(a) (b) (c)

Figure 6. Test of the proposed algorithm in typical challenging circumstances for registration: (a) large motion; (b) outliers; (c) nonrigid transformation

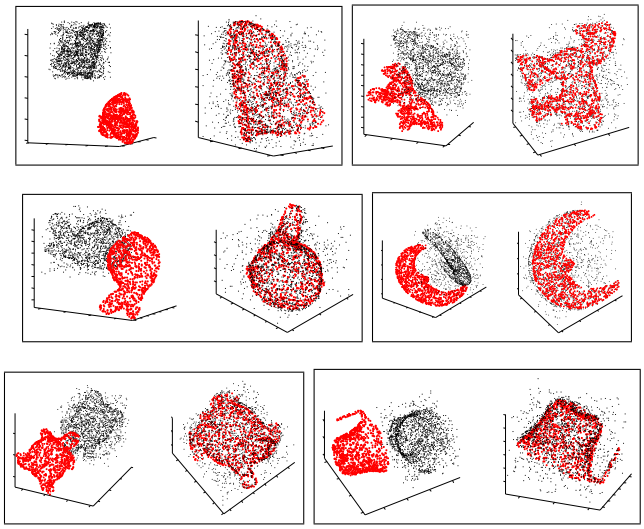
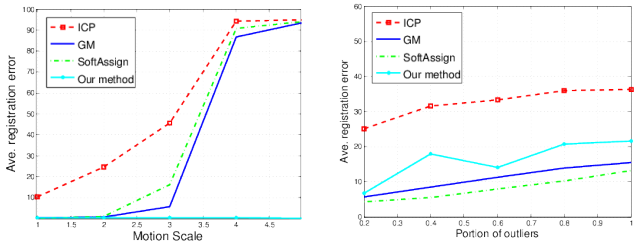


Figure 7. Some test results on KIT 3D object database



(a) (b)

Figure 8. Test of four registration algorithm on (a) different scales of motions; (b) different portion of outliers added.

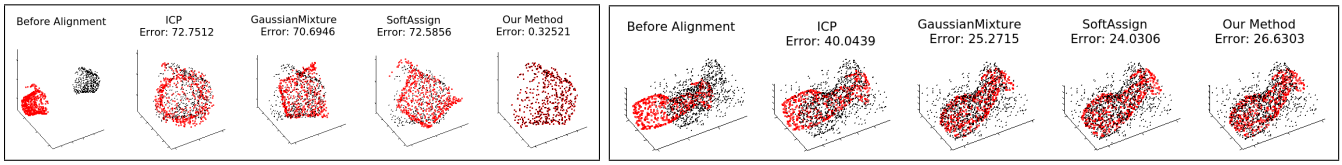
more intensive test (some test results can be seen in Figure 7). In addition, for quantitative comparison, ICP method, Gausssian Mixture(GM) and SoftAssign [3] are implemented as well. To ensure fairness, the same $SE(3)$ on-manifold optimization strategy is employed for their corresponding objective functions. Since the objects in KIT database is in triangulated mesh format, point clouds are generated by first sampling a triangle with the probability proportional to its area and then uniformly sampling a point from the selected triangle.

First, we test the robustness of four algorithms on different scales of motions. In our experiment, for motion scale i, the rotation angles of yaw, pitch and roll are $i \times [30°, 5°, 5°]$, and translations are $i \times [S_x, S_y, S_z]$, where $[S_x, S_y, S_z]$ are standard deviations of point clouds in three axes. Different motions are applied to the point cloud of each object (points cloud is sampled with size 1000) to generate a target point cloud to align with. Since we know the correspondence between the original and target point clouds, the error for each registration is computed as the average distance between every pair of corresponding points in two point clouds. The test result is plotted in Figure 8(a). We can see that four algorithms can work similarly well for small motions (scale $i = 1$). However, as i increases, the accuracy of ICP, GM and SoftAssign decrease, although GM and SoftAssign are much more robust than ICP for intermediate motions (scale $i = [2, 3]$). Our method, by contrast, performs consistently well for all scales of motions. A test example is displayed in Figure 9(a), from which we can see that the instability of ICP, GM and SoftAssign stems from the fact that they are likely to to stuck into local optimum when the motion is large (although the local optimum can be avoided by setting up many intial poses, it would take more time to guarantee that the global optimum is found).

Secondly, we test the robustness of four algorithms by adding different portion (the percentage of point cloud size) of outliers which are randomly sampled within the space around objects. The generated outliers are concatenated into the original point clouds, so the correspondence is still available and the registration error is computed in the same way as in the motion experiment. To avoid the effect of large motion, a relatively small motion (motion scale $i = 1$) is applied to all point clouds. The test result is plotted in Figure 8(b). We can see that SoftAssign is most stable for the case in which outliers are presented, GM and our method are slightly worse, and ICP is very sensitive to outlier even when the portion is small. A test example is displayed in Figure 9(b), from which we can see that except ICP, the result of other three algorithms are acceptable.

Last but never least, efficiency is a significant strength of our method, which enables it can be used for real

[3]the comparison in [12] has reported that SoftAssign and EM-ICP perform similarly, so we are not going to include EM-ICP in our experiment



(a) (b)

Figure 9. Comparison of four registration algorithms: (a) motion scale $i = 5$; (b) outlier portion$= 0.8$.

	100	200	500	1000	2000
Our method	1.172	1.489	2.162	5.126	21.165
ICP [1]	0.012	0.023	0.051	0.154	0.469
GaussianMixtures [5]	1.859	3.998	15.245	43.570	172.4
SoftAssign [3]	2.059	4.801	83.925	592.1	3812

Table I
AVERAGE EXECUTION TIME (SECONDS)

time applications. As we can see in Algorithm 1, after computing eigenvectors for kernel matrices, the complexity of computing optimal motion is linear to the size of points n. Since the complexity of fast PCA is $O(n \log n)$ [10], the overall complexity of Algorithm 1 is $O(n \log n)$. To compare the efficiency, all four algorithms are implemented in Matlab and run on the same hardware platform (usual i7 intel core laptop). Point clouds of all objects are generated with 5 different sizes (100, 200, 500, 1000, 2000), on which four algorithms are tested respectively. For each point cloud, a randomly generated motion is applied and random portion of outliers are added (to get an approximate average). Note that in this experiment we are only concerned about the running time, so the algorithms will stop when they converge even if the registration is bad. The average execution time (in seconds) of four algorithms on five set of point clouds are presented in Table I. We can see that the complexity of our algorithm is the same as ICP $O(n \log n)$, and it is much faster than SoftAssign and Gausssian Mixtures(GM) with complexity $O(n^2)$ (however, SoftAssign is usually more expensive than GM because it needs to iteratively update correspondence matrix).

CONCLUSION

We introduced a novel point cloud registration algorithm based on kernel-induced feature maps, kernel PCA and $SE(3)$ on-manifold optimization. The framework is theoretically elegant, and exhibits robustness and accuracy in fairly challenging circumstances. It is quite general and flexible to be extended to different dimensions and intra-category instances alignment . Remarkably, it outperforms most other methods in terms of efficiency.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Communitys Seventh Framework Programme FP7/2007-2013 (Specific Programme Cooperation, Theme 3, Information and Communication Technologies) under grant agreement no. 270273, Xperience.

REFERENCES

[1] P. J. Besl and H. D. Mckay, "A Method for Registration of 3-D Shapes," *PAMI*, vol. 14, no. 2, pp. 239–256, 1992.

[2] S. Rusinkiewicz and M. Levoy, "Efficient Variants of the ICP Algorithm," in *3DIM*, 2001, pp. 145–152.

[3] S. Gold, A. Rangarajan, C. Lu, and E. Mjolsness, "New Algorithms for 2D and 3D Point Matching: Pose Estimation and Correspondence," *Pattern Recognition*, vol. 31, pp. 957–964, 1997.

[4] S. Granger and X. Pennec, "Multi-scale EM-ICP: A Fast and Robust Approach for Surface Registration," in *European Conference on Computer Vision (ECCV 2002), volume 2353 of LNCS*. Springer, 2002, pp. 418–432.

[5] B. Jian and B. C. Vemuri, "Robust Point Set Registration Using Gaussian Mixture Models," *PAMI*, vol. 33, no. 8, pp. 1633–1645, 2011.

[6] R. Kondor and T. Jebara, "A Kernel between Sets of Vectors," in *ICML*, 2003.

[7] B. Schölkopf, A. Smola, E. Smola, and K.-R. Müller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural Computation*, vol. 10, pp. 1299–1319, 1998.

[8] Z. Zhang, "Iterative point matching for registration of free-form curves and surfaces," *International Journal of Computer Vision*, vol. 13, no. 2, pp. 119–152, 1994.

[9] C. J. Taylor and D. J. Kriegman, "Minimization on the Lie Group SO(3) and Related Manifolds," Yale University, Tech. Rep., 1994.

[10] A. Sharma and K. K. Paliwal, "Fast Principal Component Analysis using Fixed-point Algorithm," *Pattern Recognition Letters*, vol. 28, no. 10, pp. 1151–1155, 2007.

[11] A. Kasper, Z. Xue, and R. Dillmann, "The KIT object models database: An object model database for object recognition, localization and manipulation in service robotics," *The International Journal of Robotics Research*, May 2012.

[12] Y. Liu, "Automatic Registration of Overlapping 3D Point Clouds using Closest Points," *Image and Vision Computing*, vol. 24, no. 7, pp. 762–781, 2006.