Supersymmetric Tensor Based Higher-Order Matching

Aiping Wang[a], Ralph R. Martin[b], Zhiquan Cheng[a], Sikun Li[a], Liang Zeng[a]

[a] *School of Computer Science, National University of Defense Technology, China*
[b] *School of Computer Science and Informatics, Cardiff University, Wales, UK*

Abstract

This paper proposes a supersymmetric-tensor-based efficient higher-order matching algorithm for finding correspondences between two sets of features. Higher-order matching relies on higher-order data affinity measures, which take into account matches between more than a single pair of features at a time.

In this paper, a compact form of higher-order supersymmetric tensor (the *affinity tensor*) is used to represent higher-order data affinity, and the matching is achieved by using the rank-one approximation of the affinity tensor directly with a higher-order power method using an efficient iteration formula derived in this paper. We take advantage of supersymmetry to devise an efficient sampling strategy to create the affinity tensor.

Experiments on both synthetic and real image datasets show that our approach provides significantly improved matching of feature correspondences in comparison to other leading algorithms.

Keywords: higher-order matching; supersymmetric tensor; power method

1. Introduction

Finding correspondences between two sets of visual features (such as points of interest, edges, or regions) is a fundamental problem in many computer vision tasks, such as object recognition [1, 2], shape matching [3, 4], wide baseline stereo, and 3D reconstruction [5].

The challenging problem has attracted considerable attention for many years. It is commonly cast as an assignment problem, either a linear assignment problem, or a higher-order assignment problem. Given two sets of features P_1 and P_2, containing N_1 and N_2 features respectively: $\{f_i^1\}_{i=1}^{N_1}$, $\{f_j^2\}_{j=1}^{N_2}$, the matching problem can be seen as one of finding a set of assignments $S^* \equiv \{s_i\}_{i=1}^{n} = \{f_i^1, f_j^2\}_{i=1}^{n}$. Linear assignment matches single features in one set with single features in the other set; higher-order assignment matches multiple features in one set simultaneously with ones from the other set.

The linear assignment problem poses matching as finding a mapping $f : P_1 \rightarrow P_2$, such that the optimal assignment $S^* = \arg\max \sum_{i \in P_1} A(i, f(i))$, where $A : N_1 \times N_2 \rightarrow R$ is the *affinity matrix*: $A(i, j)$ measures the affinity between feature $i \in P_1$ and feature $j \in P_2$. As a simple example, if we define a similarity between two features (e.g. Euclidean distance for SIFT feature descriptors used in [6]), matching tries to allocate each feature in the first set to its nearest equivalent in the second feature set. The Hungarian algorithm [7] is a well-known method for solving this linear assignment problem.

Matching two feature sets just by considering similarities of single features from each set can easily fail in the presence of image ambiguities such

as repeated elements, textures or relatively uniform local appearance. To overcome this problem, correspondence constraints can be used between features to enforce feature matching consistency. For example, Leordanu and Hebert [8] consider both feature descriptors, and *distances* between pairs of features from each set to reduce the number of incorrect correspondences. Thus, if points p_1 and p_2 of set 1 are matched to points $p_1^{'}$ and $p_2^{'}$ of set 2, then the Euclidean distance between between p_1 and p_2 should be similar to the distance between $p_1^{'}$ and $p_2^{'}$. Such use of pairwise distance constraints can help to find improved correspondences in cases when the features have reduced discriminative ability. Taking affinities into account pairwise leads to a quadratic assignment problem.

In the above quadratic case, we now have an affinity matrix $A : N_1N_2 \times N_1N_2 \rightarrow R$, where $A(i,j)$ measures the compatibilities between two assignments $s_i = (f_i^1, f_{i'}^2)$ and $s_j = (f_j^1, f_{j'}^2)$, which takes into account both similarity of features and Euclidean distances. The quadratic assignment problem is to find a mapping $f : P_1 \rightarrow P_2$, as the optimal assignment $S^* = \arg\max \sum_{i,j \in P_1} A((i, f(i)), (j, f(j)))$. We can also use an affinity matrix $A^{'} : N_1N_1 \times N_2N_2 \rightarrow R$ in place of $A : N_1N_2 \times N_1N_2 \rightarrow R$ [9], in which case we now wish to find $S'^* = \arg\max \sum_{i,j \in P_1} A^{'}((i,j), (f(i), f(j)))$. Unfortunately, the quadratic assignment problem is NP-hard, but spectral relaxation techniques [8, 10] can provide good approximate solutions.

Posing feature matching problems as *higher-order* assignment problems has attracted much recent interest. Higher-order assignment further generalizes the quadratic assignment problem to include more complex constraints between features. We may define an m^{th}-order affinity measure T_m to capture

the affinity of m assignments $s_{i_1} = (f^1_{i_1}, f^2_{i'_1},)$, $\cdots$, $s_{i_m} = (f^1_{i_m}, f^2_{i'_m})$. Now, T_m encodes high-order constraints on multiple feature tuples, allowing for example scale and affine invariance, which pairwise affinity measures cannot capture.

The higher-order assignment problem is now to find a mapping $f : P_1 \rightarrow P_2$, as the optimal assignment $S^* = \arg\max \sum_{i_1,i_2,\cdots,i_m \in P_1} T((i_1, f(i_1)), \cdots, (i_m, f(i_m)))$. Compared to pairwise methods, higher-order methods further improve matching accuracy by using more powerful affinity measures to enforce matching consistency.

Zass and Sashua [11] use a hypergraph to describe higher-order affinities, in which each feature is a vertex, and the relationship among a tuple of vertices is a hyperedge involving multiple features. An m^{th} order tensor $\mathcal{T}_m$, of size $N_1N_2 \times N_1N_2 \times \cdots \times N_1N_2$, may also be used to represent the affinity information [12, 13]: the elements of $\mathcal{T}_m$ represent the affinity between two feature tuples in which each tuple includes multiple features.

While such higher-order methods can significantly improve matching accuracy, the higher-order assignment problem is again NP-hard. Zass and Sashua [11] consider a probabilistic model of soft hypergraph matching. They reduce the higher-order problem to a first-order one by marginalizing the higher-order tensor into a one dimensional probability vector. Optimal probabilistic matching results are then found via an iterative successive projection algorithm. Chertok et al. [13] treat the tensor as a joint probability of assignments, marginalize the affinity tensor down to a matrix, and find optimal soft assignments by eigendecomposition of the matrix.

Higher-order assignment problems typically require large amounts of mem-

ory and computational resources. By reducing the dimensionality of the affinity measures, the above approaches [11, 13] can efficiently match large numbers (many hundreds or more) of features. However, these approaches sparsify the affinity information to some degree, leading to a reduction in matching accuracy. When matching two feature sets with large differences in scale, the matching results can be even become unstable.

In a different approach, Duchenne et al. [12] directly operate on the affinity tensor and obtain a matching solution by computing the rank-one approximation of the tensor [14] using the power iteration method. In each iteration they adopt an ℓ-1 norm relaxation instead of the classical ℓ-2 norm relaxation, as in practice the ℓ-1 norm typically leads to a more satisfactory matching result. When matching a moderate number of features (up to a hundred), this method achieves much more accurate results.

A higher-order tensor is a natural way to represent the higher-order data affinity measures as we further discuss in Section 2.1. A higher-order data affinity measure, as one tensor element, actually quantifies the matching of multiple pairs of features, which can be defined as a *affinity function* or *potential function*. For example a third-order potential function commonly used in [12, 13] quantifies the affinity between two feature triples (i_1, j_1, k_1) and (i_2, j_2, k_2) based on measuring the similarity of the angles $(\alpha_1, \alpha_2, \alpha_3)$ and $\alpha_1^{'}, \alpha_2^{'}, \alpha_3^{'}$ formed by such two feature tuples. For example, a third-order potential item $\phi_3(s_{i_1}, s_{i_2}, s_{i_3}) = \phi_3(\{i_1, i_2\}, \{j_1, j_2\}, \{k_1, k_2\}) = \exp\{C \sum_{(l,l')} \|\alpha_l - \alpha_{l'}\|^2\}$, where $s_{i_1} = \{i_1, i_2\}, s_{i_2} = \{j_1, j_2\}, s_{i_3} = \{k_1, k_2\}$,$C$ is a const.

When using higher-order potential functions, it is typical to assume that

the value of the higher-order potential function is invariant to any permutation of the assignments [12, 13]. For example, the value of the $\phi_3(s_{i_1}, s_{i_2}, s_{i_3})$ stated above, which only calculates the total differences in corresponding angles, will not change with the order of the input assignments, like $(s_{i_1}, s_{i_2}, s_{i_3})$ or $(s_{i_3}, s_{i_2}, s_{i_1})$. Under these conditions, the affinity tensor become a *supersymmetric* tensor [15] as we further discuss in Section 2.1.

Although [12, 13] claimed that their affinity tensor is supersymmetric, they did not make full use of supersymmetry when creating the supersymmetric affinity tensor, nor did they take advantage of supersymmetry when accelerating the power iteration process.

Here, we extend the tensor based higher-order matching method of Duchenne et al. [12]. Using a supersymmetric affinity tensor, we give a more accurate and efficient higher-order matching algorithm than [12], especially for matching a moderate number of features. We make careful use of the supersymmetry of the affinity tensor, and determine correspondences by finding the rank-one approximation of the tensor using a more efficient power method.

Thus, our contributions are:

1. We show how to define a higher-order supersymmetric affinity tensor to express higher-order consistency constraints between features. It provides a compact way to express the whole tensor.
2. We give an efficient higher-order power iteration method, based on the supersymmetry of the affinity tensor. This leads to a significantly more efficient computational approach.
3. We present an efficient sampling strategy for feature tuples to create the affinity tensor. Based on the supersymmetry of the affinity tensor,

we constrain the sampling process to eliminate repetitive items and reduces the number of feature tuples to be sampled while ensuring the accuracy of the power iteration result.

4. We demonstrate a specific fourth-order potential with affine invariance, for feature matching under affine transformation.

Our experiments on both synthetic and real image data sets show that our method is more accurate and robust than prior approaches, yet has competitive computational cost.

Section 2 describes the higher-order supersymmetirc affinity tensor and the power iteration method. Considerations in using our method are discussed in Section 3. Experiments are shown in Section 4 and conclusions drawn in Section 5.

2. Supersymmetric tensor based higher-order matching

First we describe the higher-order matching problem in detail; for simplicity we assume we are matching feature point as a particular example. Suppose we are given two sets of feature points P_1 and P_2, with N_1 and N_2 features respectively. The correspondences between these two feature sets can be represented by an *assignment matrix* X of size $N_1 \times N_2$ whose elements are 0 or 1. $X(i,j) = 1$ when $i \in P_1$ matches $j \in P_2$ and $X(i,j) = 0$ otherwise. X can be row-wise vectorized to an assignment vector $\boldsymbol{x} \in \{0,1\}^{N_1 N_2}$.

Solving the higher-order matching problem is equivalent to solving the higher-order assignment problem, i.e. finding the optimal assignment vector

$\boldsymbol{x}^* =< x_{i_1}, x_{i_2}, \cdots, x_{i_N} > \in \{0,1\}^{N_1 N_2}$, satisfying

$$\boldsymbol{x}^* = \arg\max_{\boldsymbol{x}} \sum_{i_1, i_2, \cdots, i_N} T_N(i_1, i_2, \cdots, i_N) x_{i_1} x_{i_2} \cdots x_{i_N} \tag{1}$$

where i_n, stands for an assignment $(i_n^{'}, i_n^{''})$, $(i_n^{'} \in P_1, i_n^{''} \in P_2)$. The product $x_{i_1} x_{i_2} \cdots x_{i_N}$ is 1 only if all assignments $\{i_n\}_{n=1}^N$ are equal to 1, which means the features $(i_1^{'}, i_2^{'}, \cdots, i_N^{'})$ are all matched to the features $(i_1^{''}, i_2^{''}, \cdots, i_N^{''})$ separately. $T_N(i_1, i_2, \cdots, i_N)$ defines the affinity of all assignments $\{i_n\}_{n=1}^N$, and it also can be seen as the affinity measure between two ordered feature tuples: one is $(i_1^{'}, i_2^{'}, \cdots, i_N^{'})$ from P_1, the other is $(i_1^{''}, i_2^{''}, \cdots, i_N^{''})$ from P_2.

Note that we do not assume that N_1 equals N_2 in this paper. We just consider one-to-many mapping of correspondence problem, which assuming a point in P_1 is matched to exactly one point in P_2, while not true in reverse.

It can simply be made symmetric by considering P_1 matches P_2 and P_2 matches P_1 and then combining them by taking the union or intersection of matching results. Also the approach can easily be extended to allow matching certain points from P_1 to no points in P_2 using dummy nodes (see [4]). Uniqueness of matches for P_1 means that the assignment matrix X satisfies $\sum_i X(i,j) = 1$.

From Equ.(1) we can see that there are three major problems to be solved for higher-order matching algorithms:

1. How to define the affinity measure between two feature tuples, or equivalently, how to define the higher-order potential function ϕ_N;
2. How to organize and express the affinity measures T_N;
3. How to approximate the optimal higher-order assignment.

The actual form of the higher-order potential ϕ_N depends on the particular matching application; we give some examples in Section 3. We first discuss the second and third problems, which are independent of choice of potential.

2.1. Supersymmetric affinity tensor

A tensor generalises the concept of vector and matrix to higher order dimensions: a vector is a tensor of order one, and a is a tensor of order two. A higher-order tensor can be expressed as a multi-dimensional array [16]. Here we consider a higher-order supersymmetric affinity tensor, which represents higher-order affinity between multiple features using real values.

Definition 1 (Supersymmetric Tensor). *A tensor is called supersymmetric if its entries are invariant under any permutation of its indices [15].*

For example, given the third-order supersymmetric tensor $\mathcal{T}_3$, we have the relationship: $\mathcal{T}_3(i_1, i_2, i_3) = \mathcal{T}_3(i_1, i_3, i_2) = \mathcal{T}_3(i_2, i_1, i_3) = \mathcal{T}_3(i_2, i_3, i_1) = \mathcal{T}_3(i_3, i_1, i_2) = \mathcal{T}_3(i_3, i_2, i_1)$.

Now, $T_N(i_1, i_2, \cdots, i_N)$, mentioned in Equ.(1), measures the affinity of the assignments $(i_1, i_2, \cdots, i_N)$; in other words it gives a score to matching the ordered feature tuple $(i_1^{'}, i_2^{'}, \cdots, i_N^{'})$ from P_1 to the ordered feature tuple $(i_1^{''}, i_2^{''}, \cdots, i_N^{''})$ from P_2, using the value $\phi_N(i_1, i_2, \cdots, i_N)$. Often, the value of $\phi_N(i_1, i_2, \cdots, i_N)$ is invariant under permutation of its arguments $(i_1, i_2, \cdots, i_N)$—for example, see the third-order potential function ϕ_3 used in [12, 13]. Obviously, a higher-order supersymmetric tensor $\mathcal{T}$ can be used to capture this information:

Definition 2 (Supersymmetric Affinity Tensor). *Suppose we are given two feature sets P_1 and P_2, with N_1 and N_2 features respectively. An N^{th} order $I_1 \times I_2 \cdots \times I_N$, nonnegative tensor $\mathcal{T}_{\mathcal{N}}$, where $I_1 = I_2 = \cdots = I_N = \{1, 2, \cdots, N_1 N_2\}$, is the supersymmetric affinity tensor if there exist a set of indices θ_N, and an N^{th} order potential function ϕ_N, such that*

$$\mathcal{T}_N(i_1, i_2, \ldots, i_N) := \begin{cases} \phi_N(\Omega(i_1, i_2, \ldots, i_N)) & ,\forall(i_1, i_2, \ldots, i_N) \in \theta_N \\ 0 & ,\forall(i_1, i_2, \ldots, i_N) \notin \theta_N \end{cases} \tag{2}$$

where Ω stands for the arbitrary permutation of the vector. And θ_N requires that $\forall(i_1, i_2, \ldots, i_N) \in \theta_N, \forall i_p \in \{i_1, i_2, \ldots, i_N\}$ and $\forall i_q \in \{i_1, i_2, \ldots, i_N\} - \{i_p\}$, it always has $i_p \neq i_q$.

The tensor item subject to $(i_1, i_2, \ldots, i_N) \in \theta_N$ is called a potential item, while the one subject to $(i_1, i_2, \ldots, i_N) \notin \theta_N$ is called a non-potential item.

From Definition 2, we store the potential items like $\mathcal{T}_N(i_1, i_2, \ldots, i_N) = \mathcal{T}_N(sort(i_1, i_2, \ldots, i_N)), \forall(i_1, i_2, \ldots, i_N) \in \theta_N$, which means any index $(i_1, i_2, \cdots, i_N) \in \theta_N$ provides the value for $N!$ indices. There are $N! \cdot \#\{\theta_N\}$ potential items, where $\#\{\theta_N\}$ is the size of the index set θ_N, while we just need $\#\{\theta_N\}$ elements to express them. As the values of the non-potential items are all zero, there is no need to store or calculate all $(N_1 N_2)^N$ elements of the supersymmetric affinity tensor, but just the $\#\{\theta_N\}$ potential items. This property highly reduces the sampling size of the feature tuples when creating the affinity tensor as will be discussed in Section 2.4. At the same time, it can help making the following power iteration process efficient, which we will discuss in Section 2.3.

2.2. Higher-order Power Method

According to Definition 2, Equ.(1) can be expressed as:

$$\boldsymbol{x}^* = \arg\max_{\boldsymbol{x}} \sum_{i_1,i_2,\cdots,i_N} \mathcal{T}_N(i_1,i_2,\cdots,i_N)x_{i_1},x_{i_2},\cdots,x_{i_N} = \max < \mathcal{T}_N, \boldsymbol{x}^{\star N} > \tag{3}$$

where $\star$ is called the Tucker product [15], and $\boldsymbol{x} \in \{0,1\}^{N_1N_2}$.

As already noted, solving Equ.(3) is an NP-complete problem, so it is common to relax the constraints: the binary assignment vector $\boldsymbol{x} \in \{0,1\}^{N_1N_2}$ is replaced by an assignment vector $\boldsymbol{u}$ with elements taking real values in $[0,1]$. This changes the optimization problem to one of computing the rank-one approximation of the affinity tensor $\mathcal{T}_N$ [14], i.e. finding a scalar λ and a unit norm vector $\boldsymbol{u} \in \mathbb{R}^{N_1N_2}$, such that the tensor $\hat{\mathcal{T}_N} = \lambda \boldsymbol{u} \star \boldsymbol{u} \star \cdots \star \boldsymbol{u} = \boldsymbol{u}^{\star N}$ minimizes the function $f(\hat{\mathcal{T}_N}) = \|\mathcal{T}_N - \hat{\mathcal{T}_N}\|$. The final matching result is derived by discretizing the vector $\boldsymbol{u}$.

The higher-order power method is commonly used to find the rank-one tensor approximation; a version for supersymmetric tensors (S-HOPM) is given by [15]. The S-HOPM algorithm converges under the assumption of the convexity for the functional induced by the tensor [15], which is often satisfied in many practical applications. The S-HOPM algorithm is presented in Algorithm 1.

In Algorithm 1, $\overset{\mathcal{T}_N}{\star}$ is a so-called $\mathcal{T}_N$-product [15], satisfying

$$\hat{\boldsymbol{u}}^{(k)} = \mathcal{I} \overset{\mathcal{T}_N}{\star} \left(\boldsymbol{u}^{(k-1)}\right)^{\overset{\mathcal{T}_N}{\star}(N-1)} \Leftrightarrow \tag{4}$$

$$\forall i_1,\ u_{i_1}^{(k)} = \sum_{i_2,\ldots,i_N} \mathcal{T}_{i_1,i_2,\ldots,i_N} u_{i_2}^{(k-1)} u_{i_3}^{(k-1)} \cdots u_{i_N}^{(k-1)} \tag{5}$$

Step 4 normalises the assignment matrix X under the Frobenius norm. As stated in Section 2, X satisfies $\sum_i X(i,j) = 1$, so the assignment matrix in

Algorithm 1 Symmetric higher-order power iteration method

Input: Nth-order supersymmetric tensor $\mathcal{T}_N$

Output: unit-norm vector $\boldsymbol{u}$

1: Initialize $\boldsymbol{u}_0$, $k = 1$

2: **repeat**

3: $\quad \hat{\boldsymbol{u}}^{(k)} = \mathcal{I} \overset{\mathcal{T}_N}{\star} \left(\boldsymbol{u}^{(k-1)}\right) \overset{\mathcal{T}_N}{\star} (N-1)$

4: $\quad \boldsymbol{u}^{(k)} = \hat{\boldsymbol{u}}^{(k)} / \|\hat{\boldsymbol{u}}^{(k)}\|$

5: $\quad k = k + 1;$

6: **until** convergence;

Frobenius norm used in step 4 can be relaxed to a matrix under the $\mathcal{C}_2$ norm, which means the Euclidean norms of each of the N_2 columns are equal to one [12]. Moreover, Duchenne et al. [12] pointed out that instead of the $\mathcal{C}_2$ norm, with all columns with unit ℓ-2 norm, we can use the $\mathcal{C}_1$ norm, with all columns with unit ℓ-1 norm. Doing so simplifies matters: using the $\mathcal{C}_1$ norm, Equ.(5) can be rewritten:

$$\forall i_1,\ v_{i_1}^{(k)} = \sum_{i_2,\ldots,i_N} \mathcal{T}_{i_1,i_2,\ldots,i_N} 2 v_{i_1}^{(k-1)} v_{i_2}^{2_{(k-1)}} v_{i_3}^{2_{(k-1)}} \cdots v_{i_N}^{2_{(k-1)}},\ where\ \boldsymbol{u}^{(k)} = \boldsymbol{v}^{2_{(k)}}. \tag{6}$$

However, Duchenne et al. [12] failed to note that the whole iteration process can be simplified by taking advantage of supersymmetry. In [12], all tensor elements take part in the iteration process, which is unencessary. For example, given a third-order supersymmetric affinity tensor $\mathcal{T}_3$, an element $\phi_3(i,j,k)$ with index (i,j,k) and the element $\phi_3(i,k,j)$ with index (i,k,j) are the same, and so $\phi_3(i,k,j)$ can be reduced like $\phi_3(i,j,k)$.

The redundant tensor elements also lead to a further problem. There is no

guarantee in [12] that all tensor elements are provided or all tensor elements are balanced. For example, both items $\phi_3(i,j,k)$ and $\phi_3(i,k,j)$ may occur, while just one item $\phi_3(i,j,l)$, $l \neq k$ may occur amongst the elements of $\mathcal{T}_3$. Such unbalanced redundant tensor elements disturb the power iteration process, leading to incorrect results.

We solve the above problems in Sections 2.3 and 2.4.

2.3. Supersymmetric Tensor Based Higher-order Power Iteration

In this section, we give a new efficient higher-order power iteration method based on the supersymmetric affinity tensor. As many of the elements of the affinity tensor are zero non-potential items, it is more efficient to perform the power iteration by just considering potential items, then the complexity of the whole iteration process only depends on the size of the affinity items. We give an efficient power iteration formula which simplifies the iteration process by making use of supersymmetry.

We explain the idea using a third-order affinity tensor as an example. Given a potential index set θ_3 and a potential function ϕ_3, replacing all the equivalent elements in Equ.(6) by a single element, we get the following equations:

$$\begin{aligned}
&\forall(i,j,l) \in \theta_3 \ , \\
v_i^{(k)} &= \mathcal{T}_3(i,j,l) 2 v_i^{(k-1)} {v_j^{2}}^{(k-1)} {v_l^{2}}^{(k-1)} + \mathcal{T}_3(i,l,j) 2 v_i^{(k-1)} {v_l^{2}}^{(k-1)} {v_j^{2}}^{(k-1)} \\
&= 2 \cdot \mathcal{T}_3(\theta_3(i,j,l))\, 2\, v_i^{(k-1)} {v_j^{2}}^{(k-1)} {v_l^{2}}^{(k-1)} = 2 \cdot \phi_3(i,j,l)\, 2\, v_i^{(k-1)} {v_j^{2}}^{(k-1)} {v_l^{2}}^{(k-1)}
\end{aligned} \tag{7}$$

$$
\begin{aligned}
v_j^{(k)} &= \mathcal{T}_3(j,i,l)2v_j^{(k-1)}{v_i^{2}}^{(k-1)}{v_l^{2}}^{(k-1)} + \mathcal{T}_3(j,l,i)2v_j^{(k-1)}{v_l^{2}}^{(k-1)}{v_i^{2}}^{(k-1)} \\
&= 2\cdot\mathcal{T}_3(\theta_3(i,j,l))\,2\,v_j^{(k-1)}{v_i^{2}}^{(k-1)}{v_l^{2}}^{(k-1)} = 2\cdot\phi_3(i,j,l)\,2\,v_j^{(k-1)}{v_i^{2}}^{(k-1)}{v_l^{2}}^{(k-1)}
\end{aligned}
\tag{8}
$$

$$
\begin{aligned}
v_l^{(k)} &= \mathcal{T}_3(l,i,j)2v_l^{(k-1)}{v_i^{2}}^{(k-1)}{v_j^{2}}^{(k-1)} + \mathcal{T}_3(l,j,i)2v_l^{(k-1)}{v_j^{2}}^{(k-1)}{v_i^{2}}^{(k-1)} \\
&= 2\cdot\mathcal{T}_3(\theta_3(i,j,l))\,2\,v_l^{(k-1)}{v_i^{2}}^{(k-1)}{v_j^{2}}^{(k-1)} = 2\cdot\phi_3(i,j,l)\,2\,v_l^{(k-1)}{v_i^{2}}^{(k-1)}{v_j^{2}}^{(k-1)}
\end{aligned}
\tag{9}
$$

For an Nth-order supersymmetric affinity tensor, the corresponding result is:

$$
\begin{aligned}
&\forall m \in (i_1, i_2, \cdots, i_N), \\
&v_m^{(k)} = (N-1)!\cdot\phi_N(i_1,i_2,\cdots,i_N)\cdot 2v_m^{(k-1)}{v_{i_1}^{2}}^{(k-1)}\cdots{v_{m-1}^{2}}^{(k-1)}{v_{m+1}^{2}}^{(k-1)}\cdots{v_{i_N}^{2}}^{(k-1)}
\end{aligned}
\tag{10}
$$

Using Equ.(10), we can replace Algorithm 1 by the version in Algorithm 2.

This version removes each non-potential item from the iteration process, so is more efficient, and the complexity of the whole iteration process only depends on the size $\#\{\theta_N\}$ of affinities. Step 5 in Algorithm 2 makes it clear that all permutations of each potential item $\mathcal{T}_n(i_1,i_2,\cdots,i_n)$ have been considered, and have been expressed by a single potential function $\phi_n(i_1,i_2,\cdots,i_n)$. This method reduces the memory cost while keeping the accuracy. Algorithm 2 depends on all potential items. We next discuss the issue of how to sample the feature tuples to build potential items, which also determining the size $\#\{\theta_N\}$.

Many initialization schemes have been proposed for the power method [15]. We simply use positive random values to initialize u_0, which ensures that the algorithm converges to a meaningful result.

Algorithm 2 Higher-order power iteration method based on supersymmetric affinity tensor(with $\mathcal{C}_1$ norm)

Input: Nth-order supersymmetric affinity tensor $\mathcal{T}_n$

Output: unit-norm vector $\boldsymbol{u}$

Initialize $\boldsymbol{u}_0$, $k = 1$

repeat

  for all $(i_1, i_2, \cdots, i_N) \in \theta_N$ **do**

    for all $m \in (i_1, i_2, \cdots, i_N)$ **do**

    $v_m^{(k)} = (N-1)! \cdot \phi_N(i_1, i_2, \cdots, i_N) \cdot 2 v_m^{(k-1)} v_{i_1}^{2_{(k-1)}} \cdots v_{m-1}^{2_{(k-1)}} v_{m+1}^{2_{(k-1)}} \cdots v_{i_N}^{2_{(k-1)}}$

    end

    for $i = 1 : N_1$ **do**

      $v^{(k)}(((i-1) \cdot N_2 + 1) : i \cdot N_2) =$

      $\hat{v}^{(k)}(((i-1) \cdot N_2 + 1) : i \cdot N_2) / \|\hat{v}^{(k)}(((i-1) \cdot N_2 + 1) : i \cdot N_2)\|_2$

    end

  end

  $k = k + 1$;

until convergence;

note: $v^{(k)}(((i-1) \cdot N_2 + 1) : i \cdot N_2)$ means the elements of the vector $v^{(k)}$, with indices from $(i-1) \cdot N_2 + 1$ to $i \cdot N_2$.

2.4. Sampling strategy

The whole iteration process depends on the potentials, so choice of feature tuples to calculate the potentials directly influences the matching accuracy. Different sampling strategies can be chosen for different applications. We simply use random sampling for general feature correspondence problems where there is no guidance to provide a better sampling method.

Given two feature sets P_1 and P_2 including N_1 and N_2 features respec-

tively, a potential item is obtained by using two feature tuples sampled from each feature set separately. As an example, in Fig.1 we can get a feature tuple (i_1, j_1, k_1) from the left triangle, and another one (i_2, j_2, k_2) from the right triangle. Using (i_1, j_1, k_1) and (i_2, j_2, k_2) we can create a potential item of third-order tensor $\phi_3(s_1, s_2, s_3)$, with its index $s_1 = \{i_1, i_2\}, s_2 = \{i_1, j_2\}, s_3 = \{k_1, k_2\}$. In this paper the feature tuple is order-sensitive, because the feature tuple with different orders will generate different tensor items. Still take the Fig.1 as an example, if we get a feature tuple (j_1, i_1, k_1) from the left triangle, and another one (i_2, j_2, k_2) from the right triangle, we will obtain a potential item of third-order tensor $\phi_3(s_1^{'}, s_2^{'}, s_3^{'})$, with the index $s_1^{'} = \{j_1, i_2\}, s_2^{'} = \{i_1, j_2\}, s_3^{'} = \{k_1, k_2\}$. Obviously $\phi_3(s_1^{'}, s_2^{'}, s_3^{'})$ is different with the above one $\phi_3(s_1, s_2, s_3)$.

For Nth-order matching, a naive way to construct the potential items is as follows: first find all feature tuples for P_1 and P_2, as F_1 and F_2; then $\forall(f_{i_1}^1, f_{i_2}^1, \cdots, f_{i_N}^1) \in F_1$, calculating the potentials for $(f_{i_1}^1, f_{i_2}^1, \cdots, f_{i_N}^1)$ with all feature tuples in F_2. Such a naive method is very expensive.

We in practice find a better sampling method that it just needs to perform random sampling for both P_1 and P_2 when N_1 equals to N_2. The detail is as below. We generate two feature-tuple sets F_1 and F_2 for P_1 and P_2 respectively. In order to cover all features in P_1 in F_1, each time we pick up one feature as a required element, and then we randomly choose t_1 feature tuples all containing the required element. We repeat this process until all features in P_1 have been chosen once as a required element. We do the same for P_2. Now we get two feature-tuple sets F_1 and F_2, each including $N_1 t_1$ and $N_2 t_2$ feature tuples separately.

$\forall(f^1_{i_1}, f^1_{i_2}, \cdots, f^1_{i_N}) \in F_1$, we find its k nearest neighbors in F_2 to build k potential items as ϕ^k_i, $\#\{\phi^k_i\} = k$. Combining all the potential-items obtained, we can get the desired potential-item set $\theta_N = \{\phi^k_i\}^{N_1 t_1}_{i=1}$, with the size $\#\{\theta_N\} = N_1 t_1 k$. The parameters t_1, t_2 and k must be chosen according to the size of the feature sets. In practice for two point sets each with a hundred points, this approach works well when $t_1 \approx t_2 \approx 50$ and $k \approx 300$ for third-order and fourth-order matching. The sampling cost is $O(ntk \log n)$, where $n = \max(N_1, N_2), t = \max(t_1, t_2)$.

The most important thing in the process is using the supersymmetry of the affinity tensor. The definition of an N^{th}-order supersymmetric affinity tensor should satisfy:

$$\forall(i_1, i_2, \cdots, i_N), (j_1, j_2, \ldots, j_N) \in \theta_N,\ (i_1, i_2, \cdots, i_N) \neq \Omega(j_1, j_2, \cdots, j_N) \quad (11)$$

where Ω is an arbitrary permutation. Thus, we use a sampling constraint that the sets of feature tuples F_1 and F_2 obtained from the sampling process, should have no repetition, in the sense that

$$\forall(f^1_{i_1}, f^1_{i_2}, \cdots, f^1_{i_N}), (f^1_{j_1}, f^1_{j_2}, \cdots, f^1_{j_N}) \in F_1, \\ (f^1_{i_1}, f^1_{i_2}, \cdots, f^1_{i_N}) \neq \Omega(f^1_{j_1}, f^1_{j_2}, \cdots, f^1_{j_N}) \quad (12)$$

$$\forall(f^2_{i_1}, f^2_{i_2}, \cdots, f^2_{i_N}), (f^2_{j_1}, f^2_{j_2}, \cdots, f^2_{j_N}) \in F_2, \\ (f^2_{i_1}, f^2_{i_2}, \cdots, f^2_{i_N}) \neq \Omega(f^2_{j_1}, f^2_{j_2}, \cdots, f^2_{j_N}) \quad (13)$$

and Ω is arbitrary permutation. This sampling constraint eliminates overlaps that may appear in the potential items θ_N.

When N_1 is not equal to N_2 (suppose $N_1 < N_2$), we just perform the sampling strategy like the way in [12]: random sampling only for P_1 but full

sampling for P_2 (find out all the N_2^N feature tuples). However we still use the sampling constraint for feature-tuple set F_1, which makes our strategy different with the way in [12]. The process of building the tensor items is the same to the process stated above when N_1 equals to N_2. In practice such way works well when matching two feature sets with different number of features inside.

Earlier work [11, 12] also adopted random sampling, but failed to impose any constraint on the sampling process, leading to the possibility that feature tuples may include repetition. For example, for third-order matching, it is possible that a feature tuple $(f_{i_1}^1, f_{i_2}^1, f_{i_3}^1)$ may be sampled from P_1 and $(f_{i_1}^2, f_{i_2}^2, f_{i_3}^2)$ from P_2, and also a feature tuple $(f_{i_1}^1, f_{i_3}^1, f_{i_2}^1)$ sampled from P_1 and $(f_{i_1}^2, f_{i_3}^2, f_{i_2}^2)$ from P_2. That will create two tensor elements $\phi_3(s_{i_1}, s_{i_2}, s_{i_3})$ with index $(s_{i_1}, s_{i_2}, s_{i_3})$ and $\phi_3(s_{i_1}, s_{i_3}, s_{i_2})$ with index $(s_{i_1}, s_{i_3}, s_{i_2})$, which are the same. However, we just need one tensor element to express the affinity measure on the assignment group $(s_{i_1}, s_{i_2}, s_{i_3})$. This problem not only makes the potential items redundant but also affects the accuracy of the power iteration, because the numbers of each tensor element are not equal and some elements may be used more than once during the iteration progress.

A further disadvantage of [12] is that random sampling was only performed on one feature set, while all feature tuples were used from the other feature set. In contrast, our method not only reduces the sampling cost, but also improves the accuracy of the power iteration.

3. Higher-order Potentials

There is still an important problem to be considered when using our algorithm to solve feature matching problems, which is how to describe the affinity measure between two feature tuples, and how to define the higher-order potential function.

High-order potential functions take into account more stable structural properties of features on larger scales(eg.geometric structure), in contrast to pairwise potentials. Appropriately defined high-order potentials can enhance the invariance of the feature set, and in turn improve the matching accuracy.

Different higher-order potentials are appropriate for different applications. Here we introduce two higher-order potentials, one possessing similarity invariance and the other affine invariance, which can be used for many correspondence problems in computer vision. The potentials are based on Gaussian kernel which guarantee the tensor elements nonnegative and invariant to any permutation of input assignments.

First we consider a third-order similarity-invariant potential ϕ_3 linking two feature tuples, each having three features. Similarity geometrically implies that triangles formed by three points should have invariant shape under scale, rotation and translation transformations—their internal angles should not change. Thus we can define ϕ_3 in terms difference of corresponding angles of the triangles as

$$\phi_3(i,j,k) = \phi_3(\{i_1,i_2\},\{j_1,j_2\},\{k_1,k_2\}) = \exp(-1/\varepsilon^2 \sum\nolimits_{(l,l')} \|\alpha_l - \alpha_{l'}\|^2) \tag{14}$$

where $\varepsilon > 0$ is the is the kernel bandwidth, and the $\{\alpha_l\}_{l=1}^{3}$ and $\{\alpha_l^{'}\}_{l'=1'}^{3'}$ are the angles triangles formed by the feature triple (i_1, j_1, k_1) and the feature

triple (i_2, j_2, k_2), shown in Fig.1. Each point is related with one angle of which the point is the angular vertex.

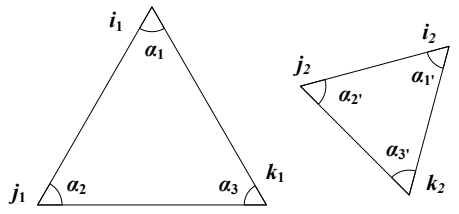


Figure 1: Third-order potential .

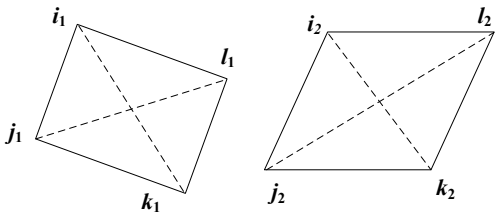


Figure 2: Fourth-order potential .

This third-order potential has been widely used [12, 13]. However, feature image matching under affine transformations is a common problem, so here we introduce a new fourth-order potential ϕ_4 which is affine-invariant, linking feature tuples with four features each. We use the affine invariance of the ratio between two closed areas to define ϕ_4 as:

$$\phi_4(i,j,k,l) = \phi_4(\{i_1,i_2\},\{j_1,j_2\},\{k_1,k_2\},\{l_1,l_2\}) \\ = \exp(-1/\varepsilon^2 \quad \textstyle\sum_{(l,l')} \|\delta_l - \delta_{l'}\|^2) \quad (15)$$

where $\{\delta_l\}_{l=1}^{4}$ and $\{\delta_l^{'}\}_{l'=1'}^{4'}$ are the ratios between the area of one triangle formed by three feature points and the area of the quadrilateral formed by all four feature points, so $\delta_1 = S_{\triangle i_1 j_1 k_1}/S_{\square i_1 j_1 k_1 l_1}$, $\delta_2 = S_{\triangle j_1 k_1 l_1}/S_{\square i_1 j_1 k_1 l_1}$, $\delta_3 = S_{\triangle i_1 k_1 l_1}/S_{\square i_1 j_1 k_1 l_1}$, $\delta_4 = S_{\triangle i_1 j_1 l_1}/S_{\square i_1 j_1 k_1 l_1}$, and similarly for the other quadrilateral.

We now use these two higher-order potentials defined above to demonstrate our algorithm.

4. Experiments

We have used synthetically generated random data as well as real images to evaluate our proposed method, and provide comparisons to other state-of-art algorithms. The first experiment uses artificial data, while the other three use real image data in three classical correspondence problems: image matching under affine transformation, image matching on deformable surfaces, and image matching under projective transformations.

We used our third-order feature matching approach in the artificial data experiment, for deformable surface matching and for image matching under projective transformation. Our fourth-order feature matching approach was applied to image matching under affine transformation.

We chose the following algorithms as a basis for comparison: the bipartite graph matching method [4] (a first-order method), the spectral method [9] (a pairwise method), a third-order tensor based method [12] and the hyper graph matching method [11], using their codes published online in each case. For fairness, all the higher-order methods, including the third-order tensor based method [12], the hyper graph matching method [11] and our method, used the same potentials.

4.1. Artificial data

First we used artificial data to verify the performance of our method quantitatively. Four tests were carried out: a rotation test, a rescaling test, a distortion test and an outlier test.

For the first three tests, we generated 50 random points in the 2D plane

for P_1, then P_2 was obtained by the following formulas:

Rotation test:

$$P_2 = R_\alpha \cdot S_\delta \cdot P_1 + N(0, 0.05),\ \alpha \in \{0, 7\pi/4\}\ \text{ step } \pi/4, \forall S_\delta \in (0.5, 1.5)$$

Scaling test:

$$P_2 = R_{\delta'} \cdot S \cdot P_1 + N(0, 0.05),\ S \in \{1, 2, 4, 6, 8\}, \forall \delta' \in (-10°, 10°)$$

Distortion test:

$$P_2 = R_{\delta'} \cdot S_\delta \cdot P_1 + N(0, \sigma),\ \sigma \in [0, 1],\ \forall \delta' \in (-10°, 10°),\ \forall S_\delta \in (0.5, 1.5)$$

where S_δ and $R_{\delta'}$ are small random disturbances of scale and rotation angle, and each time randomly generated from $(0.5, 1.5)$ and $(-10°, 10°)$ separately. N, S and α stands for the Gaussian noise, the rescaling factor and noise variance separately. In the rotation test, all points in P_1 were rotated around the straight line perpendicular to the center of the 2D plane through the same angle α. In the scaling test, all points in P_2 changed by the same scaling S. P_2 in the distortion test was generated by disturbing the position of all points in P_1 using Gaussian noise N.

For the outlier test, we generated 20 random points as P_1, and the point set P_2 was obtained by adding random outliers with appointed number sequently (from $0 - 60$ step 10, $60 - 100$ step 5) to P_1, then all combined with small random changes on scale and rotation, and also with Gaussian noise ($\sigma = 0.05$).

We compared our third order method with all four other methods stated above. Each test was executed 50 times, and we measured the matching accuracy as the number of correctly matched points divided by the total number of points that could potentially be matched. The mean accuracy

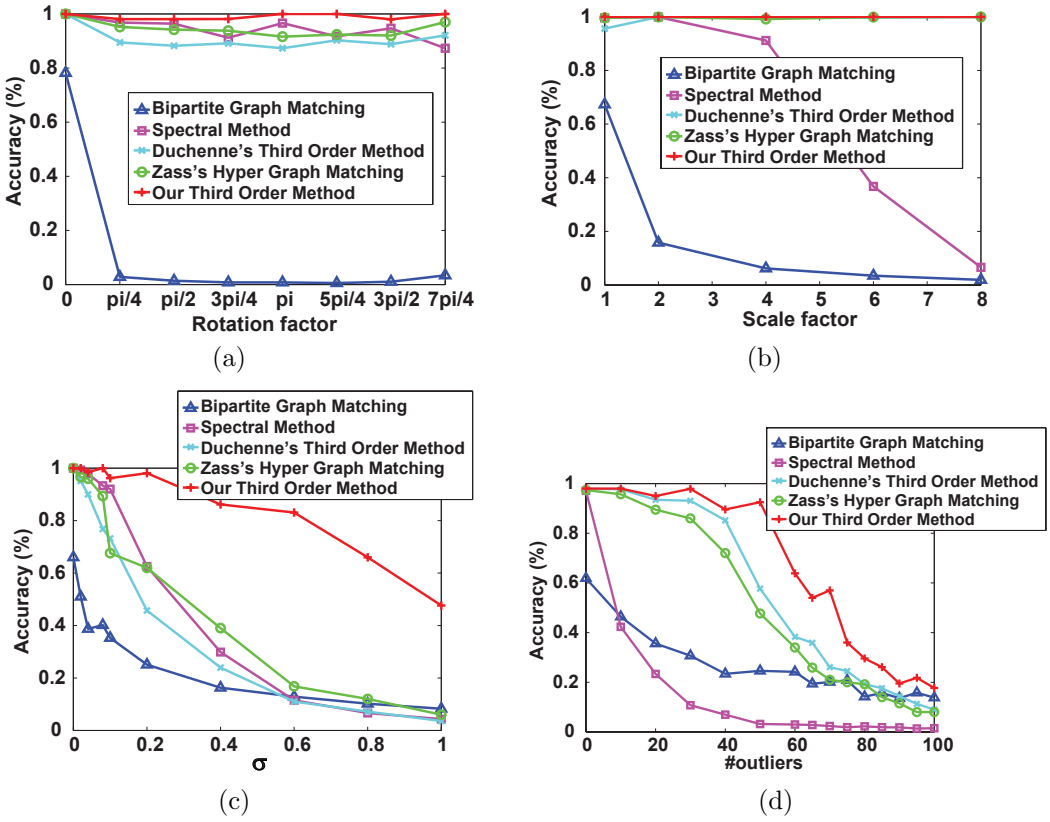


Figure 3: Fig.3(a) to 3(d), are the results of rotation test, rescaling test, distortion test and outlier test in the artificial data experiment.

over all experiments is given in Figures 3(a)–3(d).

From the figures we can see that no matter how large changes on the position (including rotation, rescaling and distortion) and outliers of P_1, our method is still able to find the correct correspondences and outperforms others. For larger rotations and rescalings, the bipartite graph matching method [4] can easily fail, while higher-order methods perform much better due to the more complex geometric constraints. However, the third order method in [12] and the hyper graph matching method do not perform well,

because the geometric relationships among elements are not established accurately. It is also clear that the spectral method [9] cannot deal with large rescalings.

In order to have a fair comparison, the size of the elements of the tensor generated in our method, [12] and [11] are kept the same in all the four tests. In rotation test, rescaling test and distortion test, for our method the total size of the feature tuples sampled from P_1 and P_2 is 5000, while the number is 150000 for [12] and 10000 for [11]. The amount of the feature tuples used in outlier test are equivalent for our method, [12] and [11].

Matching two feature sets each with 50 features inside, the average running time is around 1.8s[1] for our method, 2.6s for [12], 0.9s for [11], 0.37s for [9], 0.18s for [4]. With same tensor size while less sampling feature tuples, the higher matching accuracy show that our method are more efficient and accurate than the methods [12] and [11].

4.2. Image matching under affine transformation

Matching images related by an affine transformation is an important task. We used two publicly available image sets[2] to evaluate the matching accuracy of our method under affine transformations. The two image sets, `graf` and `wall`, each contain 6 images of a planar wall, which we numbered from 1–6. For each image set, we used the features generated from image 1 to match features in images 2 to 6 separately.

For the test image sets, we used 30 feature points detected by MSER [17]

[1] All methods were executed using Matlab code on 2.3G Core2Dou with 2GB mem.

[2] From `http://www.robots.ox.ac.uk/~vgg/data/data-aff.html`

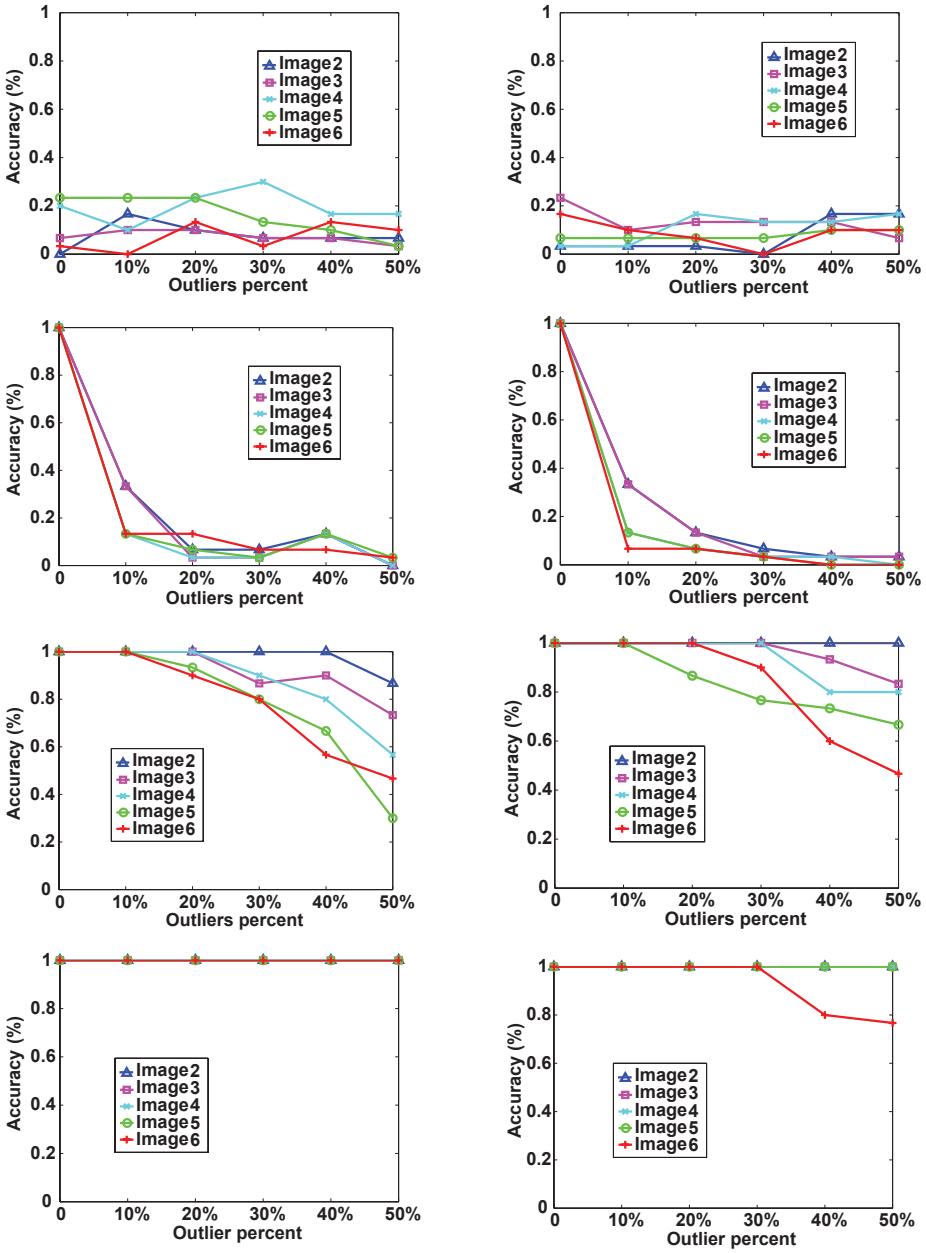
(a) (b)

Figure 4: Left: image 1 in `graf` set, right: image 1 in `wall` set. Model features detected by MSER in image 1 are shown in green; yellow points are outliers.

in the central area of image 1 to be feature set P_1 (the green points in Fig.4(b)). Then we used the the transformation matrices provided with the image sets to find the corresponding points to give the feature set P_2 for images 2–6. In order to assess the robustness of the algorithms, we manually added outliers to P_1, shown as yellow points in Fig.4(b). We successively increased the number of outliers until the ratio of outliers was 1/3.

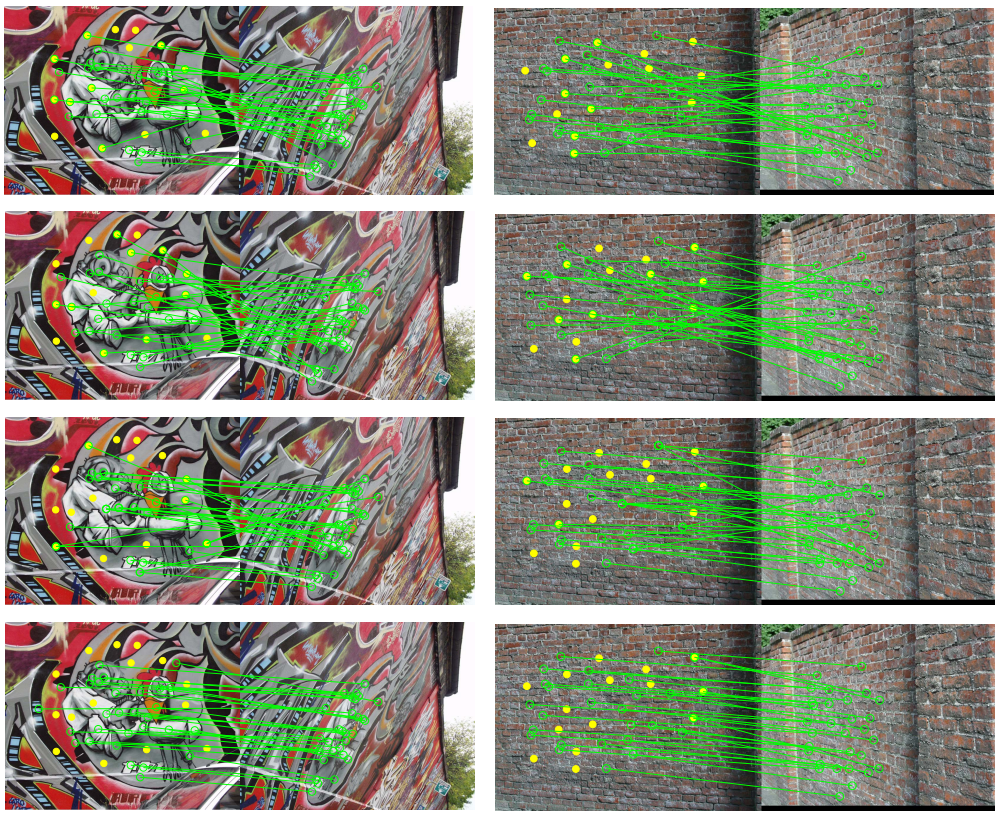
For this test, we compared our method with the spectral method [9] (a pairwise method), the tensor based method [12] and the hyper graph matching method [11], using the same fourth-order potentials for the higher-order methods. And method [12], [11] and ours all maintained the equivalent tensor size. In the test without outliers the sampling size of feature tuples used in our method is 6000, 813000 in method [12] and 12000 in method [11]. And the amounts are the same for all these three methods in the test with outliers.

In both image sets, the images are taken from quite different viewpoints, and the texture appears quite different in the different images, in both image



(a) (a)

Figure 5: Accuracy as a function of percentage of outliers. Left: `graf` set, right: `wall` set. Top to bottom: results for the spectral method [9], the hyper graph matching method [11], the tensor based method [12] and our method.



(a) (b)

Figure 6: Matching images. Left: `graf` set, right: `wall` set. Top to bottom: results of the spectral method [9], the hypergraph matching method [11], the tensor based method [12] and our method. The feature set P_1 contains 50% outliers.

sets. It is thus difficult to match features by just using an MSER detector or a SIFT detector. From the results in Fig.5, it is shown that the pairwise method SMAC [9] also fails to provide a high matching accuracy. While the higher-order methods are much accurate as higher-order structural information about the features enhances geometric consistency. Fig.5 shows that the higher-order methods work well if there are not too many outliers.

It is also clear that our method is much more stable than the other higher-

order methods. For the `graf` image set our method correctly matched the feature set P_1 to P_2 in images 2–6 up to the maximum outlier percentage. For image 6 of the `wall` set, our method still found most correct correspondences with 33.3% outliers. Though the tensor based and the hyper graph matching method [11] performed well in the absence of outliers, the matching accuracy dropped rapidly as the outlier percentage increased.

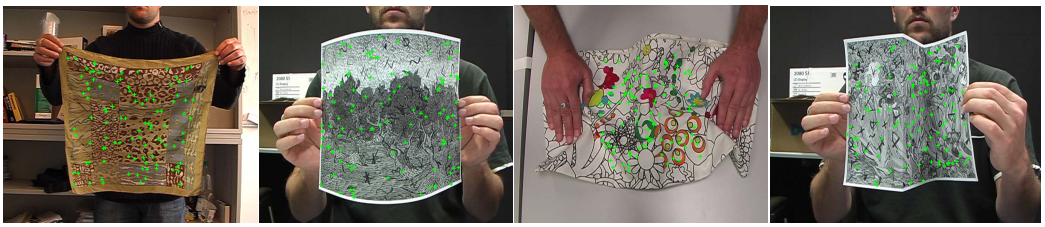
4.3. Deformable surface matching

Registration of an object subject to large non-affine deformations is also an important problem. To assess the performance of our method, we established correspondences of feature points on the deformable surface of an object using our method, the spectral method [9], the tensor based method [12] and the hyper graph matching method [11]. Again, all higher-order methods used the same third-order potentials from Section 3 and all maintained the equivalent tensor size.

We chose four deformable surface image sets[3] including four different deformable objects: a cloth, a bending paper, a cushion and a creased paper. The surfaces of the cloth and the bending paper underwent relatively smooth deformations, while the surfaces of the cushion and the creased paper included sharp folds.

From each image set we randomly chose six frames and each of them undergoes large deformation changes. We randomly chose 100 correspondent points on each deformable surface to be the features, from the provided ground truth. In each image set we chose the features on a frame, with the

[3]From `http://cvlab.epfl.ch/data/dsr/`



Frame No.299 Frame No.314 Frame No.213 Frame No.375

Figure 7: Left to right: point sets on the surface of a piece of cloth, a piece of smoothly bending paper, a creased cushion and a piece of creased paper. The frame number is below each image.

largest change on the surface and shown in Fig.7, as P_1 and matched them with the features in the other five frames.

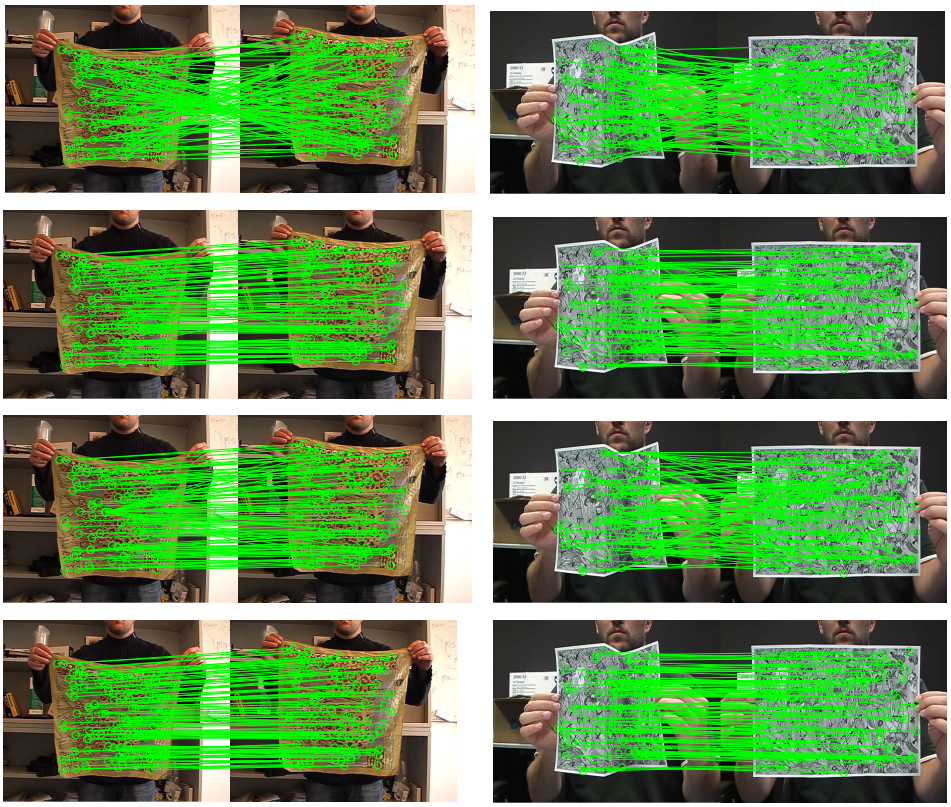
Table 1: Error rate of deformable surface matching.

Dataset	*cloth*					*bending paper*				
Matching frames	*F88-F299*	*F107-F299*	*F120-F299*	*F258-F299*	*F305-F299*	*F262-F314*	*F275-F314*	*F280-F314*	*F289-F314*	*F301-F314*
Our method	0	0	0	0	0	0	0	0	0	0
[11]	2%	2%	2%	2%	2%	23%	9%	23%	18%	14%
[12]	25%	41%	30%	30%	42%	71%	67%	59%	55%	50%
[9]	93%	83%	77%	94%	90%	93%	98%	99%	93%	96%

Table 2: Error rate of deformable surface matching.

Dataset	*cushion*					*creased paper*				
Matching frames	*F144-F213*	*F156-F213*	*F165-F213*	*F172-F213*	*F188-F213*	*F309-F375*	*F320-F375*	*F330-F375*	*F340-F375*	*F352-F375*
Our method	0	0	0	0	0	0	0	0	0	0
[11]	8%	10%	2%	6%	5%	22%	10%	11%	2%	6%
[12]	61%	65%	59%	49%	30%	80%	74%	77%	68%	46%
[9]	95%	92%	93%	93%	92%	98%	94%	96%	92%	88%

According to the provided ground truth, the matching accuracy is given as the number of correctly matched points divided by the total number of points that could potentially be matched. The final results of all methods,



(a) (b)

Figure 8: Matching images. Left: results for the cloth set F258 matching with F299, right: creased paper set, F309 matching with F375. Rows: from top to bottom, results for the spectral method [9], the hyper graph matching method [11], the tensor based method [12] and our method.

on the four image sets, are given in Tables 1 and 2. We also pick up some results of matching in Fig.8. It is clear to see that our method matched all the features on large deformable surface,while others did not. In this test the sampling size of feature tuples used in our method is 20000, 1010000 in method [12] and 40000 in method [11] separately. And the average running time, for matching two feature sets each with 100 features inside, is around 8s for our method, 13s for [12], 6.5s for [11], 5s for [9]. As above it can be seen

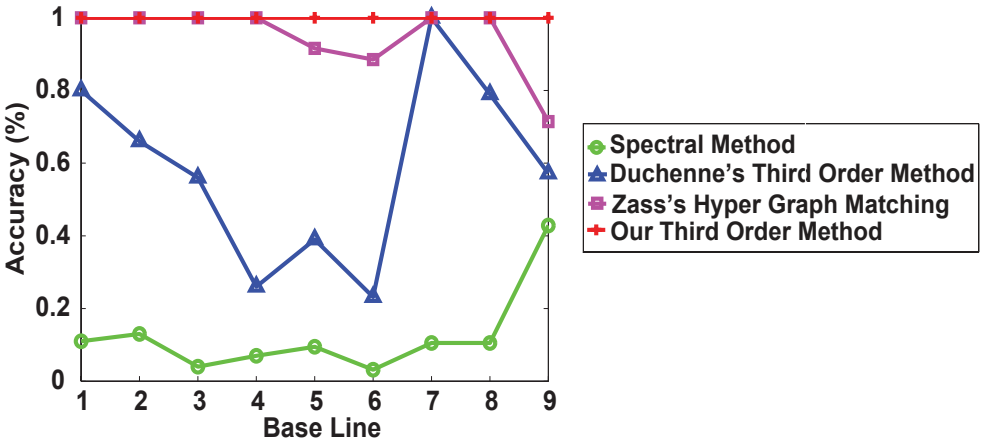


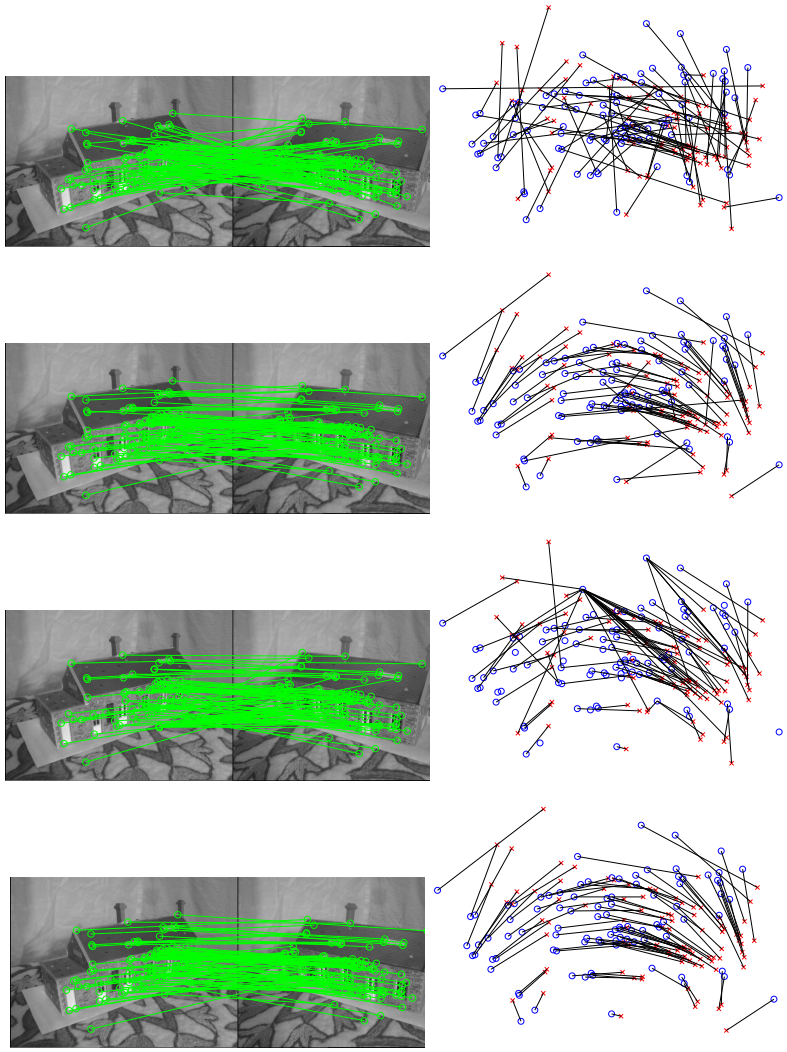
Figure 9: Accuracy as a function of the base line.

that our method is much efficient and accurate than the other high-order methods [11, 12].

4.4. Image matching under projective transformation

Finally, we used the house data set[4] to assess performance of the matching algorithms under perspective transformations. The data includes ten different viewpoints under projective transformation, numbered from 0–9. The Harris corners in every frame are labeled in advance. We used the Harris corners in image 0 as feature set P_1, and those in other images as P_2. Because ground truth correspondences are provided, we can directly compute the matching accuracy of the algorithm, as the number of correctly matched points divided by the total number of points that could potentially be matched. All the higher-order methods used the same third-order potentials from Section 3, and all kept the equivalent tensor size.

[4] Fromhttp://www.robots.ox.ac.uk/~vgg/data/data-mview.html



(a) (b)

Figure 10: Matching images 0 and 6 for the house data. Column: top to bottom, results of the spectral method [9], the hyper graph matching method [11], the tensor based method [12] and our method. Row: right side are are scatter diagrams of the matching results related with each method. The red $\times$ and blue $\circ$ represent two graphs respectively.

The larger the baseline separating the images, the larger the relative deformation, and hence the more difficult the matching. From Fig. 9, it is clear to see that our method is most stable as the baseline increases. Even

there are only seven Harris corners in each frame, it is still difficult to match because of the large viewpoint differential. The average size of sampling feature tuples for all these nine baseline matches are 13161, 643305 and 26322 for our method, method [12] and method [11].

5. Conclusion

This paper has given an efficient higher-order matching algorithm based on the supersymmetric affinity tensor, based on an efficient power iteration method. Our main contributions are as follows. First, we give a higher-order supersymmetric affinity tensor with a compact form to express higher-order consistency constraints of features. Secondly, we derive an efficient higher-order power iteration method, which makes significant savings by taking advantage of supersymmetry. We also give an efficient sampling strategy for feature tuples to create the affinity tensor. Finally, we give a fourth-order potential which possesses affine invariance. Our experiments on both synthetic and real image data sets show that our method has improved matching performance compared to state-of-the-art approaches.

References

[1] D. G. Lowe, Object recognition from local scale-invariant features, in: Proceedings of the International Conference on Computer Vision, pp. 1150–1157.

[2] A. C. Berg, T. L. Berg, J. Malik, Shape matching and object recognition using low distortion correspondence, in: IEEE Conference on Computer Vision and Pattern Recognition, pp. 26–33.

[3] Y. Zheng, D. Doermann, Robust point matching for nonrigid shapes by preserving local neighborhood structures, IEEE Trans. Pattern Anal. Mach. Intell. 28 (2006) 2006.

[4] S. Belongie, J. Malik, J. Puzicha, Shape matching and object recognition using shape contexts, IEEE Trans. Pattern Anal. Mach. Intell. 24 (2002) 509–522.

[5] R. I. Hartley, A. Zisserman, Multiple View Geometry in Computer Vision, Cambridge University Press, second edition, 2004.

[6] D. G. Lowe, Distinctive image features from scale-invariant keypoints, Int. J. Comput. Vision 60 (2004) 91–110.

[7] J. Munkres, Algorithms for the Assignment and Transportation Problems, Journal of the Society for Industrial and Applied Mathematics 5 (1957) 32–38.

[8] M. Leordeanu, M. Hebert, A spectral technique for correspondence problems using pairwise constraints, in: International Conference of Computer Vision, pp. 1482 – 1489.

[9] T. Cour, P. Srinivasan, J. Shi, Balanced graph matching, in: Advanced in Neural Information Processing Systems.

[10] S. Umeyama, An eigendecomposition approach to weighted graph matching problems, IEEE Trans. Pattern Anal. Mach. Intell. 10 (1988) 695–703.

[11] R. Zass, A. Shashua, Probabilistic graph and hypergraph matching, in: IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–8.

[12] O. Duchenne, F. Bach, I. Kweon, J. Ponce, A tensor-based algorithm for high-order graph matching, pp. 1980–1987.

[13] M. Chertok, Y. Keller, Efficient high order matching, IEEE Trans. Pattern Anal. Mach. Intell. 32 (2010) 2205–2215.

[14] L. D. Lathauwer, B. D. Moor, J. Vandewalle, On the best rank-1 and rank-(r1,r2,. . .,rn) approximation of higher-order tensors, SIAM J. Matrix Anal. Appl. 21 (2000) 1324–1342.

[15] E. Kofidis, P. A. Regalia, On the best rank-1 approximation of higher-order supersymmetric tensors, SIAM J. Matrix Anal. Appl. 23 (2002) 863–884.

[16] T. G. Kolda, B. W. Bader, Tensor decompositions and applications, SIAM Review 51 (2009) 455–500.

[17] J. Matas, O. Chum, M. Urban, T. Pajdla, Robust wide-baseline stereo from maximally stable extremal regions, Image and Vision Computing 22 (2004) 761 – 767.