Neurocomputing 142 (2014) 529–541

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Linear time Principal Component Pursuit and its extensions using ℓ_1 filtering

Risheng Liu [a,b], Zhouchen Lin [c,*], Zhixun Su [d], Junbin Gao [e]

[a] Faculty of Electronic Information and Electrical Engineering, Dalian University of Technology, Dalian, China
[b] School of Software Technology, Dalian University of Technology, Dalian, China
[c] Key Laboratory of Machine Perception (MOE), School of EECS, Peking University, Beijing, China
[d] School of Mathematical Sciences, Dalian University of Technology, Dalian, China
[e] School of Computing and Mathematics, Charles Sturt University, Bathurst, NSW 2795, Australia

ARTICLE INFO

Article history:
Received 23 September 2013
Received in revised form
4 March 2014
Accepted 10 March 2014
Communicated by E.W. Lang
Available online 27 May 2014

Keywords:
Robust principal component analysis
Principal component Pursuit
ℓ_1 minimization
Subspace learning
Incremental learning

ABSTRACT

In the past decades, exactly recovering the intrinsic data structure from corrupted observations, which is known as Robust Principal Component Analysis (RPCA), has attracted tremendous interests and found many applications in computer vision and pattern recognition. Recently, this problem has been formulated as recovering a low-rank component and a sparse component from the observed data matrix. It is proved that under some suitable conditions, this problem can be exactly solved by Principal Component Pursuit (PCP), i.e., minimizing a combination of nuclear norm and ℓ_1 norm. Most of the existing methods for solving PCP require Singular Value Decompositions (SVDs) of the data matrix, resulting in a high computational complexity, hence preventing the applications of RPCA to very large scale computer vision problems. In this paper, we propose a novel algorithm, called ℓ_1 filtering, for *exactly* solving PCP with an $O(r^2(m+n))$ complexity, where $m \times n$ is the size of data matrix and r is the rank of the matrix to recover, which is supposed to be much smaller than m and n. Moreover, ℓ_1 filtering is *highly parallelizable*. It is the first algorithm that can *exactly* solve a nuclear norm minimization problem in *linear time* (with respect to the data size). As a preliminary investigation, we also discuss the potential extensions of PCP for more complex vision tasks encouraged by ℓ_1 filtering. Experiments on both synthetic data and real tasks testify the great advantage of ℓ_1 filtering in speed over state-of-the-art algorithms and wide applications in computer vision and pattern recognition societies.

1. Introduction

Robustly recovering the intrinsic low-dimensional structure of high-dimensional visual data, which is known as Robust Principal Component Analysis (RPCA), plays a fundamental role in various computer vision and pattern recognition tasks, such as face image alignment and processing, structure from motion, background modeling, photometric stereo and texture representation (see e.g., [1–5], to name just a few).

Through the years, a large number of approaches have been proposed for solving this problem. The representative works include [2,6–10]. The main limitation of above-mentioned methods is that there is no theoretical guarantee for their performance. Recently, the advances in compressive sensing have led to increasing interests in considering RPCA as a problem of exactly recovering a low-rank matrix $\mathbf{L}_0$ (with the size $m \times n$) from corrupted observations $\mathbf{M} = \mathbf{L}_0 + \mathbf{S}_0$, where $\mathbf{S}_0$ is known to be sparse [1,11]. Its mathematical model is as follows:

$$\min_{\mathbf{L},\mathbf{S}} \operatorname{rank}(\mathbf{L}) + \lambda \|\mathbf{S}\|_{\ell_0} \quad \text{s.t. } \mathbf{M} = \mathbf{L} + \mathbf{S}, \tag{1}$$

where $\|\cdot\|_{\ell_0}$ is the ℓ_0 norm of a matrix, i.e., the number of nonzero entries in the matrix.

Unfortunately, (1) is known to be NP-hard. So Candés et al. [11] proposed using Principal Component Pursuit (PCP) to solve (1), which is to replace the rank function and the ℓ_0 norm with the nuclear norm (which is the sum of the singular values of a matrix, denoted as $\|\cdot\|_*$) and the ℓ_1 norm (which is the sum of the absolute values of the entries), respectively. More specifically, PCP is to solve the following convex problem instead:

$$\min_{\mathbf{L},\mathbf{S}} \|\mathbf{L}\|_* + \lambda \|\mathbf{S}\|_{\ell_1} \quad \text{s.t. } \mathbf{M} = \mathbf{L} + \mathbf{S}. \tag{2}$$

The work in [11] also *rigorously proved* that under fairly general conditions and $\lambda = 1/\sqrt{\max(m,n)}$, PCP can *exactly recover* the low-

* Corresponding author.
E-mail addresses: rsliu@dlut.edu.cn (R. Liu), zlin@pku.edu.cn (Z. Lin), zxsu@dlut.edu.cn (Z. Su), jbgao@csu.edu.au (J. Gao).

http://dx.doi.org/10.1016/j.neucom.2014.03.046
0925-2312/

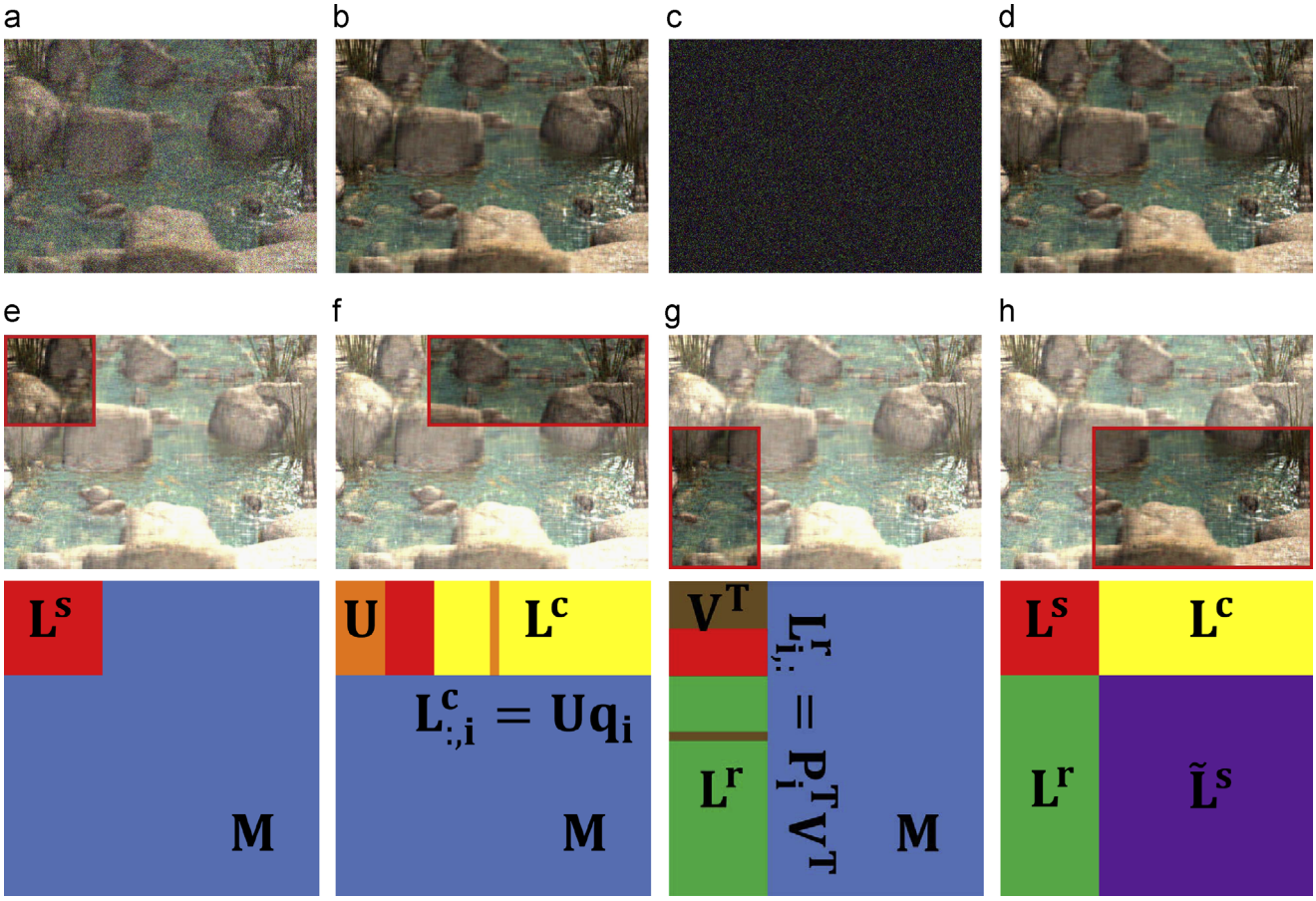

Fig. 1. Illustration of the proposed ℓ_1 filtering method. A large observed data matrix $\mathbf{M}$ (a) is the sum of a low-rank matrix $\mathbf{L}_0$ (b) and a sparse matrix $\mathbf{S}_0$ (c). The method first recovers a seed matrix (a submatrix of $\mathbf{L}_0$) $\mathbf{L}^s$ (e). Then the submatrices $\mathbf{L}^c$ (f) and $\mathbf{L}^r$ (g) can be recovered by column and row filtering, respectively, where $\mathbf{U}$ and $\mathbf{V}^T$ are the column space and row space of $\mathbf{L}^s$, respectively. Then the complement matrix $\tilde{\mathbf{L}}^s$ (h) can be represented by $\mathbf{L}^s$, $\mathbf{L}^c$ and $\mathbf{L}^r$. Finally, we obtain the computed low-rank matrix $\mathbf{L}^*$ (d), which is identical to $\mathbf{L}_0$ with an overwhelming probability. (a) M, (b) L_0, (c) S_0, (d) L^*, (e) L^s, (f) L^c, (g) L^r and (h) $\tilde{\mathbf{L}}^s$.

rank matrix $\mathbf{L}_0$ (namely the underlying low-dimensional structure) with an overwhelming probability, i.e., the difference of the probability from 1 decays exponentially when the matrix size increases. This theoretical analysis makes PCP distinct from previous methods for RPCA.

1.1. Main idea and our contribution

In this paper, we address the large-scale RPCA problem and propose a truly linear cost method to solve the PCP model (2) when the data size is very large while the target rank is relatively small. Such kind of data is ubiquitous in computer vision and pattern recognition. Our algorithm fully utilizes the properties of low-rankness. The main idea is to apply PCP to a randomly selected submatrix of the original noisy matrix and compute a low-rank submatrix. Using this low-rank submatrix, the true low-rank matrix can be estimated efficiently, where the low-rank submatrix is part of it.

Specifically, our method consists of two steps (illustrated in Fig. 1). The first step is to recover a submatrix[1] $\mathbf{L}^s$ (Fig. 1 (e)) of $\mathbf{L}_0$. We call this submatrix the seed matrix because all other entries of $\mathbf{L}_0$ can be further calculated by this submatrix. $\mathbf{L}^s$ is computed by sampling a submatrix of $\mathbf{M}$ and solving a small scale PCP on this submatrix. The second step is to use the seed matrix to recover two submatrices $\mathbf{L}^c$ and $\mathbf{L}^r$ (Fig. 1(f) and (g)), which are on the same rows and columns as $\mathbf{L}^s$ in $\mathbf{L}_0$, respectively. They are recovered by minimizing the ℓ_1 distance from the subspaces spanned by the columns and rows of $\mathbf{L}^s$, respectively. Hence we call this step ℓ_1 filtering. Since the rank of $\mathbf{L}^s$ equals to that of $\mathbf{L}_0$ (at an overwhelming probability), the remaining part $\tilde{\mathbf{L}}^s$ (Fig. 1(h)) of $\mathbf{L}_0$ can be represented by $\mathbf{L}^s$, $\mathbf{L}^c$ and $\mathbf{L}^r$, using the generalized Nystrom method [12]. We do not multiply $\mathbf{L}^s$, $\mathbf{L}^c$ and $\mathbf{L}^r$ together to form $\tilde{\mathbf{L}}^s$ explicitly, thus saving both storage and computation greatly.

As analyzed in Section 3.3.1, our method is of linear cost with respect to the data size. Besides the advantage of linear time cost, the proposed algorithm is also highly parallel: the columns of $\mathbf{L}^c$ and the rows of $\mathbf{L}^r$ can be recovered fully independently. We also prove that under suitable conditions, our method can *exactly* recover the underling low-rank matrix $\mathbf{L}_0$ with an overwhelming probability. To our best knowledge, this is the first algorithm that can *exactly* solve a nuclear norm minimization problem in *linear time*.

As a preliminary investigation, we further discuss potential extensions of PCP for more complex visual analysis tasks, such as online subspace learning and subspace clustering. Our goal is to show the readers some basic ideas on using the mechanism of ℓ_1 filtering to extend PCP for more complex computer vision and pattern recognition tasks.

The rest of this paper is organized as follows. In Section 2, we briefly review previous algorithms for solving PCP. We then present our ℓ_1 filtering algorithm in Section 3. We provide

[1] Note that the "submatrix" here does not necessarily mean that we have to choose consecutive rows and columns from $\mathbf{M}$.

extensive experiments to verify the efficacy of ℓ_1 filtering in Section 4. After that, we discuss some potential extensions and variations of PCP using ℓ_1 filtering in Section 5. Finally, we conclude our paper in Section 6.

2. Previous works

In this section, we review some previous algorithms for solving PCP. The existing solvers can be roughly divided into three categories: classic convex optimization, factorization and compressed optimization.

For small sized problems, PCP can be reformulated as a semidefinite program and then be solved by standard interior point methods. However, this type of methods cannot handle even moderate scale matrices due to their $O(n^6)$ complexity in each iteration. So people turned to first-order algorithms, such as the dual method [13], the Accelerated Proximal Gradient (APG) method [13] and the Alternating Direction Method (ADM) [14], among which ADM is the most efficient. All these methods require solving the following kind of subproblem in each iteration:

$$\min_{\mathbf{A}} \eta\|\mathbf{A}\|_*+\frac{1}{2}\|\mathbf{A}-\mathbf{W}\|_F^2, \tag{3}$$

where $\|\cdot\|_F$ is the Frobenius norm. Cai et al. [15] proved that the above problem has a closed form solution:

$$\mathbf{A}=\mathbf{U}\mathcal{S}_\eta(\mathbf{\Sigma})\mathbf{V}^T, \tag{4}$$

where $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ is SVD of $\mathbf{W}$ and $\mathcal{S}_\eta(x)=\mathrm{sgn}(x)\max(|x|-\eta,0)$ is the soft shrinkage operator. Therefore, these methods all require computing SVDs for some matrices, resulting in $O(mn\min(m,n))$ complexity, where $m\times n$ is the matrix size.

As the most expensive computational task required by solving (2) is to perform SVD, Lin et al. [14] adopted partial SVD [16] to reduce the complexity at each iteration to $O(rmn)$, where r is the target rank. However, such a complexity is still too high for very large data sets. Drineas et al. [17] developed a fast Monte Carlo algorithm, named Linear-Time SVD (LTSVD), which can be used for solving SVDs approximately (also see [18]). The main drawback of LTSVD is that it is less accurate than the standard SVD as it uses random sampling. So the whole algorithm needs more iterations to achieve the same accuracy. As a consequence, the speed performance of LTSVD quickly deteriorates when the target rank increases (see Fig. 2). Actually, even adopting LTSVD the whole algorithm is still quadratic w.r.t. the data size because it still requires matrix–matrix multiplication in each iteration.

To address the scalability issue of solving large-scale PCP problems, Shen et al. [19] proposed a factorization based method, named Low-Rank Matrix Fitting (LMaFit). This approach represents the low-rank matrix as a product of two matrices and then minimizes over the two matrices alternately. Although they do not require nuclear norm minimization (hence the SVDs), the convergence of the proposed algorithm is not guaranteed as the corresponding problem is non-convex. Moreover, both the matrix–matrix multiplication and the QR decomposition based rank estimation technique require $O(rmn)$ complexity. So this method does not essentially reduce the complexity.

Inspired by compressed optimization, the work in [20] proposed reducing the problem scale by Random Projection. However, as this method solved a modified optimization model with introduced auxiliary variables, the theoretical guarantee for the original PCP model is no longer applicable to it. Moreover, the need to introduce additional constraint to the problem slows down the convergence. And actually, the complexity of this method is also $O(pmn)$, where $p\times m$ is the size of the random projection matrix and $p>r$. So this method is still not of linear complexity with respect to the matrix size.

3. The ℓ_1 filtering algorithm

Given an observed data matrix $\mathbf{M}\in\mathbb{R}^{m\times n}$, which is the sum of a low-rank matrix $\mathbf{L}_0$ and a sparse matrix $\mathbf{S}_0$, PCP is to recover $\mathbf{L}_0$ from $\mathbf{M}$. What our approach differs from traditional ones is that the underlying low-rank matrix $\mathbf{L}_0$ is reconstructed from a seed matrix. As explained in Section 1.1, our ℓ_1 filtering algorithm consists of two steps: first recovering a seed matrix, second performing ℓ_1 filtering on the corresponding rows and columns of the data matrix. Below we provide details of these two steps.

3.1. Seed matrix recovery

Suppose that the target rank r is very small compared with the data size: $r\ll\min(m,n)$. We first randomly sample an $(s_r r)\times(s_c r)$ submatrix $\mathbf{M}^s$ from $\mathbf{M}$, where $s_r>1$ and $s_c>1$ are the row and column oversampling rates, respectively. Then the submatrix $\mathbf{L}^s$ of the underlying matrix $\mathbf{L}_0$ can be recovered by solving a small sized PCP problem:

$$\min_{\mathbf{L}^s,\mathbf{S}^s}\|\mathbf{L}^s\|_*+\tilde{\lambda}\|\mathbf{S}^s\|_{\ell_1}\quad \text{s.t. } \mathbf{M}^s=\mathbf{L}^s+\mathbf{S}^s, \tag{5}$$

e.g., using ADM, where $\tilde{\lambda}=1/\sqrt{\max(s_r r,s_c r)}$.

By Theorem 1.1 in [11], the seed matrix $\mathbf{L}^s$ can be exactly recovered from $\mathbf{M}^s$ with an overwhelming probability when s_r and s_c increase. In fact, by that theorem s_r and s_c should be chosen at

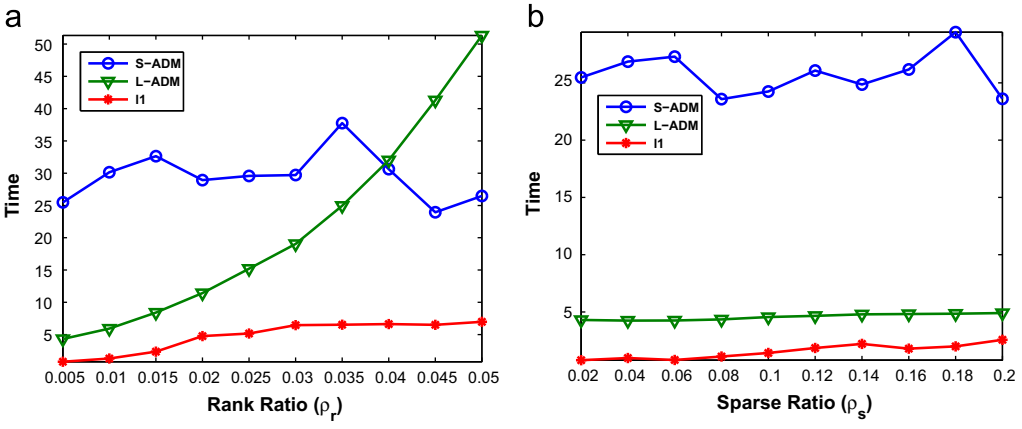

Fig. 2. Performance of the S-ADM, L-ADM and ℓ_1 filtering under different rank ratios ρ_r and sparsity ratios ρ_s, where the matrix size is 1000×1000. The x-axis represents the rank ratio (a) or sparsity ratio (b). The y-axis represents the CPU time (in seconds).

the scale of $O(\ln^2 r)$. For the experiments conducted in this paper, whose r's are very small, we simply choose $s_c = s_r = 10$.

3.2. ℓ_1 Filtering

For ease of illustration, we assume that $\mathbf{M}^s$ is the top left $(s_r r) \times (s_c r)$ submatrix of $\mathbf{M}$. Then accordingly $\mathbf{M}$, $\mathbf{L}_0$ and $\mathbf{S}_0$ can be partitioned into:

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}^s & \mathbf{M}^c \\ \mathbf{M}^r & \tilde{\mathbf{M}}^s \end{bmatrix}, \quad \mathbf{L}_0 = \begin{bmatrix} \mathbf{L}^s & \mathbf{L}^c \\ \mathbf{L}^r & \tilde{\mathbf{L}}^s \end{bmatrix}, \quad \mathbf{S}_0 = \begin{bmatrix} \mathbf{S}^s & \mathbf{S}^c \\ \mathbf{S}^r & \tilde{\mathbf{S}}^s \end{bmatrix}. \tag{6}$$

Since $\mathrm{rank}(\mathbf{L}_0) = \mathrm{rank}(\mathbf{L}^s) = r$, there must exist matrices $\mathbf{Q}$ and $\mathbf{P}$, such that

$$\mathbf{L}^c = \mathbf{L}^s\mathbf{Q} \quad \text{and} \quad \mathbf{L}^r = \mathbf{P}^T\mathbf{L}^s. \tag{7}$$

As $\mathbf{S}_0$ is sparse, so are $\mathbf{S}^c$ and $\mathbf{S}^r$. Therefore, $\mathbf{Q}$ and $\mathbf{P}$ can be found by solving the following problems:

$$\min_{\mathbf{S}^c,\mathbf{Q}} \|\mathbf{S}^c\|_{\ell_1} \quad \text{s.t. } \mathbf{M}^c = \mathbf{L}^s\mathbf{Q} + \mathbf{S}^c, \tag{8}$$

and

$$\min_{\mathbf{S}^r,\mathbf{P}} \|\mathbf{S}^r\|_{\ell_1} \quad \text{s.t. } \mathbf{M}^r = \mathbf{P}^T\mathbf{L}^s + \mathbf{S}^r, \tag{9}$$

respectively. The above two problems can be easily solved by ADM.

With $\mathbf{Q}$ and $\mathbf{P}$ computed, $\mathbf{L}^c$ and $\mathbf{L}^r$ are obtained as (7). Again by $\mathrm{rank}(\mathbf{L}_0) = \mathrm{rank}(\mathbf{L}^s) = r$, the generalized Nystrom method [12] gives

$$\tilde{\mathbf{L}}^s = \mathbf{L}^r(\mathbf{L}^s)^\dagger\mathbf{L}^c, \tag{10}$$

where $(\mathbf{L}^s)^\dagger$ is the Moore–Penrose pseudo-inverse of $\mathbf{L}^s$.

In real computation, as the SVD of $\mathbf{L}^s$ is readily available when solving (5), due to the singular value thresholding operation (4), it is more convenient to reformulate (8) and (9) as

$$\min_{\mathbf{S}^c,\tilde{\mathbf{Q}}} \|\mathbf{S}^c\|_{\ell_1}, \quad \text{s.t. } \mathbf{M}^c = \mathbf{U}^s\tilde{\mathbf{Q}} + \mathbf{S}^c, \tag{11}$$

and

$$\min_{\mathbf{S}^r,\tilde{\mathbf{P}}} \|\mathbf{S}^r\|_{\ell_1}, \quad \text{s.t. } \mathbf{M}^r = \tilde{\mathbf{P}}^T(\mathbf{V}^s)^T + \mathbf{S}^r, \tag{12}$$

respectively, where $\mathbf{U}^s\mathbf{\Sigma}^s(\mathbf{V}^s)^T$ is the skinny SVD of $\mathbf{L}^s$ obtained from (4) in the iterations. Such a reformulation has multiple advantages. First, as $(\mathbf{U}^s)^T\mathbf{U}^s = (\mathbf{V}^s)^T\mathbf{V}^s = \mathbf{I}$, it is unnecessary to compute the inverse of $(\mathbf{U}^s)^T\mathbf{U}^s$ and $(\mathbf{V}^s)^T\mathbf{V}^s$ when updating $\tilde{\mathbf{Q}}$ and $\tilde{\mathbf{P}}$ in the iterations of ADM. Second, computing (10) also becomes easy if one wants to form $\tilde{\mathbf{L}}^s$ explicitly because now

$$\tilde{\mathbf{L}}^s = \tilde{\mathbf{P}}^T(\mathbf{\Sigma}^s)^{-1}\tilde{\mathbf{Q}}. \tag{13}$$

By the relationship (7), we can rewrite (10) as

$$\tilde{\mathbf{L}}^s = \mathbf{L}^r(\mathbf{L}^s)^\dagger\mathbf{L}^c = \mathbf{P}^T\mathbf{L}^s(\mathbf{L}^s)^\dagger\mathbf{L}^s\mathbf{Q} = \mathbf{P}^T\mathbf{L}^s\mathbf{Q}$$

which is based on a property of pseudo-inverse, i.e., $\mathbf{L}^s(\mathbf{L}^s)^\dagger\mathbf{L}^s = \mathbf{L}^s$. In this view, $\mathbf{P}^T\mathbf{L}^s\mathbf{Q}$ is reduced to $\mathbf{P}^T\mathbf{L}^r$ or $\mathbf{L}^c\mathbf{Q}$. This not only speeds up computation, but also avoids the numerical instability of (13) as this formulation requires to invert the singular values of $\mathbf{L}^s$.

To make the algorithm description complete, we sketch in Algorithm 1 the ADM for solving (11) and (12), which are both of the following form:

$$\min_{\mathbf{E},\mathbf{Z}} \|\mathbf{E}\|_{\ell_1} \quad \text{s.t. } \mathbf{X} = \mathbf{A}\mathbf{Z} + \mathbf{E}, \tag{14}$$

where $\mathbf{X}$ and $\mathbf{A}$ are known matrices and $\mathbf{A}$ has orthonormal columns, i.e., $\mathbf{A}^T\mathbf{A} = \mathbf{I}$. The ADM for (14) is to minimize the following augmented Lagrangian function:

$$\|\mathbf{E}\|_{\ell_1} + \langle \mathbf{Y}, \mathbf{X} - \mathbf{A}\mathbf{Z} - \mathbf{E}\rangle + \frac{\beta}{2}\|\mathbf{X} - \mathbf{A}\mathbf{Z} - \mathbf{E}\|_F^2, \tag{15}$$

with respect to $\mathbf{E}$ and $\mathbf{Z}$, respectively, by fixing other variables, and then update the Lagrange multiplier $\mathbf{Y}$ and the penalty parameter β.[2]

Algorithm 1. Solving (14) by ADM.

Input: $\mathbf{X}$ and $\mathbf{A}$.
Initialize: Set $\mathbf{E}_0$, $\mathbf{Z}_0$ and $\mathbf{Y}_0$ to zero matrices. Set $\varepsilon > 0$, $\rho > 1$ and $\overline{\beta} \gg \beta_0 > 0$.
while $\|\mathbf{X} - \mathbf{A}\mathbf{Z}_k - \mathbf{E}_k\|_{\ell_\infty} / \|\mathbf{X}\|_{\ell_\infty} \geq \varepsilon$ **do**
  Step 1: Update $\mathbf{E}_{k+1} = \mathcal{S}_{\beta_k^{-1}}(\mathbf{X} - \mathbf{A}\mathbf{Z}_k + \mathbf{Y}_k/\beta_k)$, where $\mathcal{S}$ is the soft-thresholding operator [15].
  Step 2: Update $\mathbf{Z}_{k+1} = \mathbf{A}^T(\mathbf{X} - \mathbf{E}_{k+1} + \mathbf{Y}_k/\beta_k)$.
  Step 3: Update $\mathbf{Y}_{k+1} = \mathbf{Y}_k + \beta_k(\mathbf{X} - \mathbf{A}\mathbf{Z}_{k+1} - \mathbf{E}_{k+1})$ and $\beta_{k+1} = \min(\rho\beta_k, \overline{\beta})$.
end while

Note that it is easy to see that (11) and (12) can also be solved in full parallelism as the columns and rows of $\mathbf{L}^c$ and $\mathbf{L}^r$ can be computed independently, thanks to the decomposability of the problems. So the recovery of $\mathbf{L}^c$ and $\mathbf{L}^r$ is very efficient if one has a parallel computing platform, such as a general purpose graphics processing unit (GPU).

3.3. The complete algorithm

Now we are able to summarize in Algorithm 2 our ℓ_1 filtering method for solving PCP, where steps 3 and 4 can be done in parallel.

Algorithm 2. Solving PCP (2) by ℓ_1 filtering.

Input: Observed data matrix $\mathbf{M}$.
Step 1: Randomly sample a submatrix $\mathbf{M}^s$.
Step 2: Solve the small sized PCP (5), e.g., by ADM, to recover the seed matrix $\mathbf{L}^s$.
Step 3: Reconstruct $\mathbf{L}^c$ by solving (11).
Step 4: Reconstruct $\mathbf{L}^r$ by solving (12).
Step 5: Represent $\tilde{\mathbf{L}}^s$ by (13).
Output: Low-rank matrix $\mathbf{L}$ and sparse matrix $\mathbf{S} = \mathbf{M} - \mathbf{L}$.

3.3.1. Complexity analysis

Now we analyze the computational complexity of the proposed Algorithm 2. For the step of seed matrix recovery, the complexity of solving (5) is only $O(r^3)$. For the ℓ_1 filtering step, it can be seen that the complexity of solving (11) and (12) is $O(r^2 n)$ and $O(r^2 m)$, respectively. So the total complexity of this step is $O(r^2(m+n))$. As the remaining part $\tilde{\mathbf{L}}^s$ of $\mathbf{L}_0$ can be represented by $\mathbf{L}^s$, $\mathbf{L}^c$ and $\mathbf{L}^r$, using the generalized Nystrom method [12],[3] and recall that $r \ll \min(m,n)$, we conclude that the overall complexity of Algorithm 2 is $O(r^2(m+n))$, which is only of linear cost with respect to the data size.

It should be emphasized that though accelerated SVD decomposition (e.g., Linear Time SVD (LTSVD) [17]) can speed up classic convex optimization algorithms (e.g. ADM) for solving nuclear norm minimization, the matrix–matrix multiplication still cannot

[2] The ADM for solving PCP follows the same methodology. As a reader can refer to [14,21] for details, we omit the pseudo-code for using ADM to solve PCP.

[3] Of course, if we explicitly form $\tilde{\mathbf{L}}^s$ then this step costs no more than rmn complexity. Compared with other methods, our rest computations are all of $O(r^2(m+n))$ complexity at the most, while those methods all require at least $O(rmn)$ complexity *in each iteration*, which results from matrix–matrix multiplication.

be avoid at each iteration, thus the complexity is at least quadratic [22], which is higher than that of our ℓ_1 filtering. This has also been illustrated in our experimental part (see Section 4.1.1).

3.4. Discussions

This subsection discusses the exact recoverability and target rank estimation details of ℓ_1 filtering framework.

3.4.1. Exact recoverability of ℓ_1 filtering

The exact recoverability of $\mathbf{L}_0$ using our ℓ_1 filtering method consists of two factors. First, exactly recovering $\mathbf{L}^s$ from $\mathbf{M}^s$. Second, exactly recovering $\mathbf{L}^c$ and $\mathbf{L}^r$. If all $\mathbf{L}^s$, $\mathbf{L}^c$, and $\mathbf{L}^r$ can be exactly recovered, $\mathbf{L}_0$ is exactly recovered.

The exact recoverability of $\mathbf{L}^s$ from $\mathbf{M}^s$ is guaranteed by Theorem 1.1 of [11]. When s_r and s_c are sufficiently large, the chance of success is overwhelming.

To analyze the exact recoverability of $\mathbf{L}^c$ and $\mathbf{L}^r$, we first observe that it is equivalent to the exact recoverability of $\mathbf{S}^c$ and $\mathbf{S}^r$. By multiplying annihilation matrices $\mathbf{U}^{s,\perp}$ and $\mathbf{V}^{s,\perp}$ to both sides of (11) and (12), respectively, we may recover $\mathbf{S}^c$ and $\mathbf{S}^r$ by solving

$$\min_{\mathbf{S}^c} \|\mathbf{S}^c\|_{\ell_1} \quad \text{s.t. } \mathbf{U}^{s,\perp}\mathbf{M}^c = \mathbf{U}^{s,\perp}\mathbf{S}^c, \tag{16}$$

and

$$\min_{\mathbf{S}^r} \|\mathbf{S}^r\|_{\ell_1} \quad \text{s.t. } \mathbf{M}^r(\mathbf{V}^{s,\perp})^T = \mathbf{S}^r(\mathbf{V}^{s,\perp})^T, \tag{17}$$

respectively. If the oversampling rates s_c and s_r are large enough, we are able to choose $\mathbf{U}^{s,\perp}$ and $\mathbf{V}^{s,\perp}$ that are close to Gaussian random matrices. Then we may apply the standard theory in compressed sensing [23] to conclude that if the oversampling rates s_c and s_r are large enough and $\mathbf{S}^c$ and $\mathbf{S}^r$ are sparse enough,[4] $\mathbf{S}^c$ and $\mathbf{S}^r$ can be exactly recovered with an overwhelming probability.

We also present an example in Fig. 1 to illustrate the exact recoverability of ℓ_1 filtering. We first truncate the SVD of a 1024×768 image "Water"[5] to get a matrix of rank 30 (Fig. 1(b)). The observed image (Fig. 1(a)) is obtained from Fig. 1(b) by adding large noise to 30% of the pixels uniformly sampled at random (Fig. 1(c)). Suppose we have the top-left 300×300 submatrix as the seed (Fig. 1(e)), the low-rank image (Fig. 1(d)) can be exactly recovered by ℓ_1 filtering. Actually, the relative reconstruction errors in $\mathbf{L}^*$ are only 7.03×10^{-9}.

3.4.2. Target rank estimation

The above analysis and computation are all based on a known value of the target rank r. For some applications, we could have an estimate on r. For example, for the background modeling problem [1], the rank of the background video should be very close to one as the background hardly changes; and for the photometric stereo problem [4] the rank of the surface normal map should be very close to three as the normals are three dimensional vectors. However, the rank r of the underlying matrix might not always be known. So we have to provide a strategy to estimate r.

As we assume that the size $m' \times n'$ of submatrix $\mathbf{M}^s$ is $(s_r r) \times (s_c r)$, where s_r and s_c should be sufficiently large in order to ensure the exact recovery of $\mathbf{L}^s$ from $\mathbf{M}^s$, after we have computed $\mathbf{L}^s$ by solving (5), we may check whether

$$m'/r' \geq s_r \quad \text{and} \quad n'/r' \geq s_c \tag{18}$$

are satisfied, where r' is the rank of $\mathbf{L}^s$. If yes, $\mathbf{L}^s$ is accepted as a seed matrix. Otherwise, it implies that $m' \times n'$ may be too small with respect to the target rank r. Then we may increase the size of the submatrix to $(s_r r') \times (s_c r')$ and repeat the above procedure until (18) is satisfied or

$$\max(m'/m, n'/n) > 0.5. \tag{19}$$

We require (19) because the speed advantage of our l_1 filtering algorithm will quickly lost beyond this size limit (see Fig. 2). If we have to use a submatrix whose size should be greater than $(0.5m) \times (0.5n)$, then the target rank should be comparable to the size of data, hence breaking our low-rank assumption. In this case, we may resort to the usual method to solve PCP.

Of course, we may sample one more submatrix to cross validate the estimated target rank r. When r is indeed very small, such a cross validation is not a big overhead.

4. Experimental results

In this section, we present experiments on both synthetic data and real vision problems (structure from motion and background modeling) to test the performance of ℓ_1 filtering. All the experiments are conducted and timed on the same PC with an AMD Athlon® II X4 2.80 GHz CPU that has 4 cores and 6 GB memory, running Windows 7 and Matlab (Version 7.10).

4.1. Comparison results for solving PCP

We first test the performance of ℓ_1 filtering on solving PCP (2). The experiments are categorized into the following three classes:

- Compare with classic numerical algorithms on randomly generated low-rank and sparse matrices.
- Compare with factorization based algorithm on recovering either randomly generated or deterministic low-rank matrix from its sum with a random sparse matrix.
- Compare with compressed optimization based algorithm on recovering randomly generated low-rank and sparse matrices.

In the experiments synthetic data, we generate random test data in the following way: an $m \times m$ observed data matrix $\mathbf{M}$ is synthesized as the sum of a low-rank matrix $\mathbf{L}_0$ and a sparse matrix $\mathbf{S}_0$. The rank r matrix $\mathbf{L}_0$ is generated as a product of two $m \times r$ matrices whose entries are i.i.d. Gaussian random variables with zero mean and unit variance. The matrix $\mathbf{S}_0$ is generated as a sparse matrix whose support is chosen uniformly at random, and whose p non-zero entries are i.i.d. uniformly in $[-500, 500]$. The rank ratio and the sparsity ratio are denoted as $\rho_r = r/m$ and $\rho_s = p/m^2$, respectively.

4.1.1. ℓ_1 Filtering vs. classic convex optimization

Firstly, we compare our approach with ADM on the whole matrix, which we call the standard ADM, and its variation, which uses LTSVD[6] for solving the partial SVD, hence we call the LTSVD ADM (L-ADM). We choose these two approaches because the Standard ADM (S-ADM)[7] is known to be the most efficient classic convex optimization algorithm to solve PCP exactly and L-ADM has a linear time cost in solving SVD.[8] For L-ADM, in each time to compute the partial SVD we uniformly oversample $5r$ columns of

[4] As the analysis in the compressed sensing theories is qualitative and the bounds are actually pessimistic, copying those inequalities here is not very useful. So we omit the mathematical descriptions for brevity.

[5] The image is available at http://www.petitcolas.net/fabien/watermarking/image_database/.

[6] The Matlab code of LTSVD is available in FPCA package at http://www.columbia.edu/~sm2756/FPCA.htm.

[7] The Matlab code of S-ADM is provided by the authors of [14] and all the parameters in this code are set to their default values.

[8] However, L-ADM is still of $O(rmn)$ complexity as it involves matrix–matrix multiplication in each iteration. See also Section 2.

Table 1
Comparison among the S-ADM, L-ADM and ℓ_1 filtering (ℓ_1 for short) on the synthetic data. We present CPU time (in seconds) and the numerical accuracy of tested algorithms. $\mathbf{L}_0$ and $\mathbf{S}_0$ are the ground truth and $\mathbf{L}^*$ and $\mathbf{S}^*$ are the solution computed by different methods. For ℓ_1 filtering, we report its CPU time as $t=t_1+t_2$, where t, t_1 and t_2 are the time for total computation, seed matrix recovery and ℓ_1 filtering, respectively.

Size	Method	RelErr	rank($\mathbf{L}^*$)	$\|\mathbf{L}^*\|_*$	$\|\mathbf{S}^*\|_{\ell_0}$	$\|\mathbf{S}^*\|_{\ell_1}$	Time
2000	rank($\mathbf{L}_0$) = 20, $\|\mathbf{L}_0\|_* = 39,546$, $\|\mathbf{S}_0\|_{\ell_0} = 40,000$, $\|\mathbf{S}_0\|_{\ell_1} = 998,105$						
	S-ADM	1.46×10^{-8}	20	39,546	39,998	998,105	84.73
	L-ADM	4.72×10^{-7}	20	39,546	40,229	998,105	27.41
	ℓ_1	1.66×10^{-8}	20	39,546	40,000	998,105	**5.56**=**2.24**+**3.32**
5000	rank($\mathbf{L}_0$) = 50, $\|\mathbf{L}_0\|_* = 249,432$, $\|\mathbf{S}_0\|_{\ell_0} = 250,000$, $\|\mathbf{S}_0\|_{\ell_1} = 6,246,093$						
	S-ADM	7.13×10^{-9}	50	249,432	249,995	6,246,093	1093.96
	L-ADM	4.28×10^{-7}	50	249,432	250,636	6,246,158	195.79
	ℓ_1	5.07×10^{-9}	50	249,432	250,000	6,246,093	**42.34**=**19.66**+**22.68**
10,000	rank($\mathbf{L}_0$) = 100, $\|\mathbf{L}_0\|_* = 997,153$, $\|\mathbf{S}_0\|_{\ell_0} = 1,000,000$, $\|\mathbf{S}_0\|_{\ell_1} = 25,004,070$						
	S-ADM	1.23×10^{-8}	100	997,153	1,000,146	25,004,071	11,258.51
	L-ADM	4.26×10^{-7}	100	997,153	1,000,744	25,005,109	1301.83
	ℓ_1	2.90×10^{-10}	100	997,153	1,000,023	25,004,071	**276.54**=**144.38**+**132.16**

the data matrix without replacement.[9] For all methods in comparison, the stopping criterion is $\|\mathbf{M}-\mathbf{L}^*-\mathbf{S}^*\|_F/\|\mathbf{M}\|_F \le 10^{-7}$.

Table 1 shows the detailed comparison among the three methods, where RelErr $= \|\mathbf{L}^*-\mathbf{L}_0\|_F/\|\mathbf{L}_0\|_F$ is the relative error to the true low-rank matrix $\mathbf{L}_0$. It is easy to see that our ℓ_1 filtering approach has the highest numerical accuracy and is also much faster than the S-ADM and L-ADM. Although L-ADM is faster than the S-ADM, its numerical accuracy is the lowest among the three methods because it is probabilistic.

We also present in Fig. 2 the CPU times of the three methods when the rank ratio ρ_r and sparsity ratio ρ_s increase, respectively. The observed matrices are generated using the following parameter settings: $m=1000$, vary ρ_r from 0.005 to 0.05 with fixed $\rho_s=0.02$ and vary ρ_s from 0.02 to 0.2 with fixed $\rho_r=0.005$. It can be seen from Fig. 2(a) that L-ADM is faster than the S-ADM when $\rho_r<0.04$. However, the computing time of L-ADM grows quickly when ρ_r increases. It even becomes slower than the S-ADM when $\rho_r\ge0.04$. This is because LTSVD cannot guarantee the accuracy of partial SVD in each iteration. So its number of iterations is larger than that of the S-ADM. In comparison, the time cost of our ℓ_1 filtering method is much less than the other two methods for all the rank ratios. However, when ρ_r further grows the advantage of ℓ_1 filtering will be lost quickly, because ℓ_1 filtering has to compute the PCP on the $(s_r r)\times(s_c r)=(10r)\times(10r)$ submatrix $\mathbf{M}^s$. In contrast, Fig. 2(b) indicates that the CPU time of these methods grows very slowly with respect to the sparsity ratio.

4.1.2. ℓ_1 Filtering vs. factorization method

We then compare the proposed ℓ_1 filtering with a factorization method (i.e., LMaFit[10]) on solving (2). To test the ability of these algorithms in coping with corruptions with large magnitude, we multiply a scale $\boldsymbol{\sigma}$ to the sparse matrix, i.e., $\mathbf{M}=\mathbf{L}_0+\sigma\mathbf{S}_0$. We fix other parameters of the data ($m=1000$, $r=0.01m$ and $\rho_s=0.01$) and vary the scale parameter σ from 1 to 10 to increase the magnitude of the sparse errors.

The computational comparisons are presented in Fig. 3. Besides the CPU time and relative error, we also measure the quality of the recovered $\mathbf{L}^*$ by its maximum difference (MaxDif) and average difference (AveDif) to the true low-rank matrix $\mathbf{L}_0$, which are respectively defined as MaxDif $=\max(|\mathbf{L}^*-\mathbf{L}_0|)$ and AveDif $=(\sum_{ij}|\mathbf{L}^*-\mathbf{L}_0|)/m^2$. One can see that the performance of LMaFit dramatically decreases when $\sigma\ge3$. This experiment suggests that the factorization method fails when the sparse matrix dominates the low-rank one in magnitude. This is because a sparse matrix with large magnitudes makes rank estimation difficult or impossible for LMaFit. Without a correct rank, the low-rank matrix cannot be recovered exactly. In comparison, our ℓ_1 filtering always performs well on the test data.

In the following, we consider the problem of recovering *deterministic* low-rank matrix from corruptions. We generate an $m\times m$ "checkerboard" image (see Fig. 4), whose rank is 2, and corrupt it by adding 10% impulsive noise to it. The corruptions (nonzero entries of the sparse matrix) are sampled uniformly at random. The image size m ranges from 1000 to 5000 with an increment of 500.

The results for this test are shown in Fig. 4, where the first image is the corrupted checkerboard image, the second image is recovered by LMaFit and the third by ℓ_1 filtering. A more complete illustration for this test can be seen from Fig. 4(d), where the CPU time corresponding to all tested data matrix sizes is plotted. It can be seen that the images recovered by LMaFit and ℓ_1 filtering are visually comparable in quality. The speeds of these two methods are very similar when the data size is small, while ℓ_1 filtering runs much faster than LMaFit when the matrix size increases. This concludes that our approach has significant speed advantage over the factorization method on large scale data sets.

4.1.3. ℓ_1 Filtering vs. compressed optimization

Now we compare ℓ_1 filtering with a compressed optimization method (i.e., Random Projection[11]). This experiment is to study the performance of these two methods with respect to the rank of the matrix and the data size. The parameters of the test matrices are set as follows: $\rho_s=0.01$, ρ_r varying from 0.05 to 0.15 with fixed $m=1000$, and m varying from 1000 to 5000 with fixed $\rho_r=0.05$. For the dimension of the projection matrix (i.e., p), we set it as $p=2r$ for all the experiments.

As shown in Fig. 5, in all cases the speed and the numerical accuracy of ℓ_1 filtering are always much higher than those of random projection.

[9] As there is no general suggestion for setting this parameter, we experimentally set it as $5r$ and found that such an oversampling rate is important for ensuring the numerical accuracy of L-ADM at high probability.

[10] The Matlab code of LMaFit is provided by the authors of [19] and all the parameters in this code are set to their default values.

[11] The Matlab code of Random Projection is provided by the author of [20] and all the parameters in this code are set to their default values.

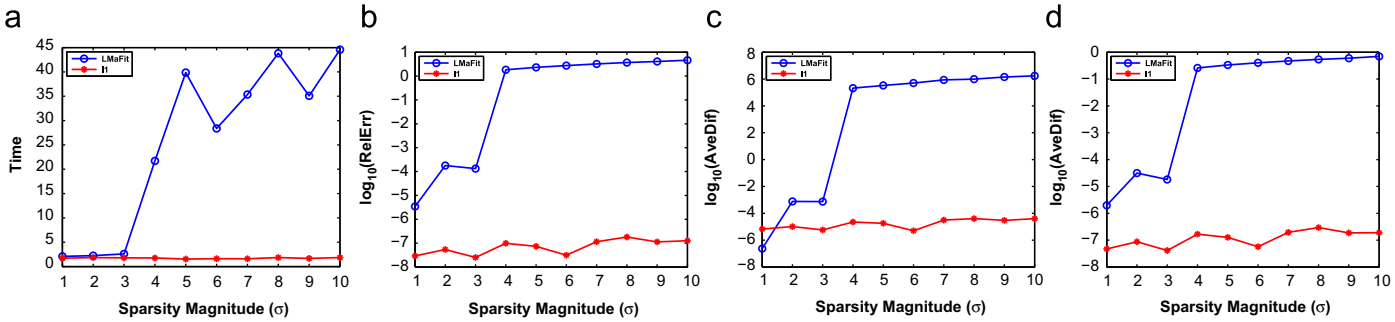

Fig. 3. Performance of LMaFit and ℓ_1 filtering under different sparsity magnitudes ($\sigma \in [1, 10]$). The x-axes represent the sparsity magnitudes and the y-axes represent the CPU time (in seconds) (a), "RelErr" (b), "MaxDif" (c) and "AveDif" (d) in log scale, respectively.

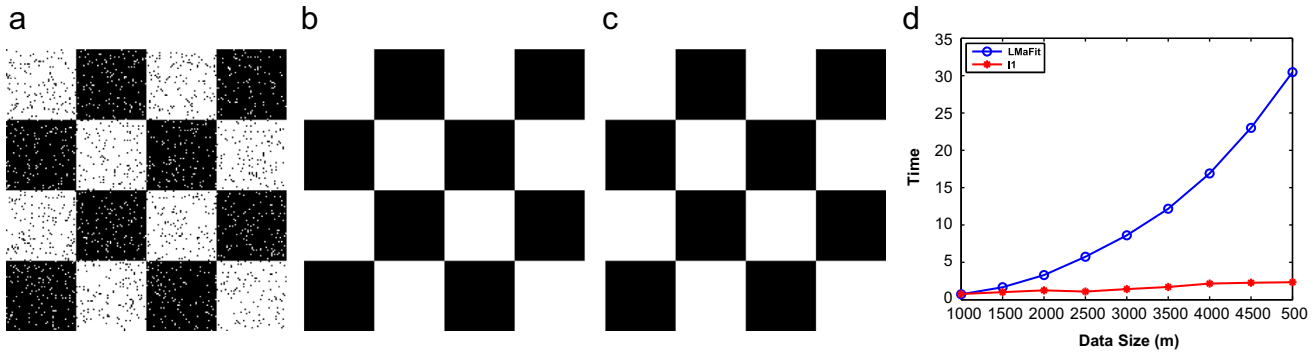

Fig. 4. Recovery results for "checkerboard". (a) Is the image corrupted by 10% impulsive noise. (b) Is the image recovered by LMaFit. (c) Is the image recovered by ℓ_1 filtering. (d) CPU time (in seconds) vs. data size ($m \in [1000, 5000]$). (a) Corrupted, (b) LMaFit, (c) ℓ_1 and (d) Time.

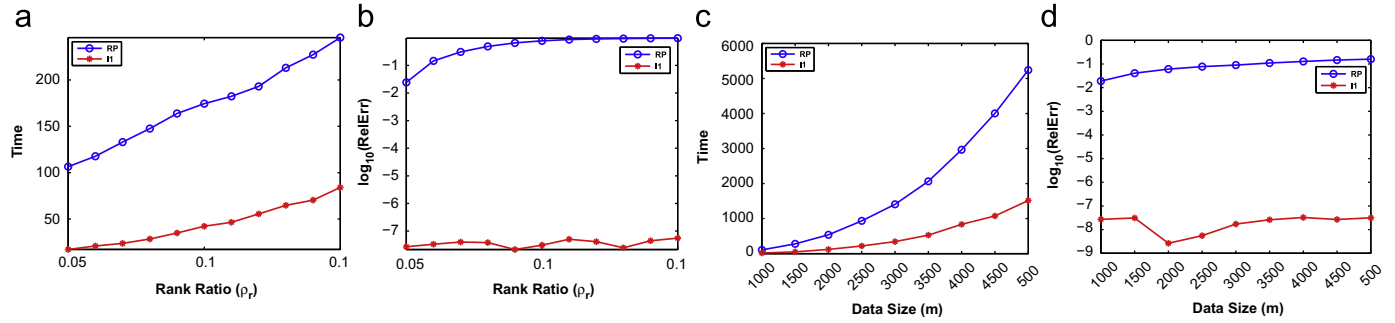

Fig. 5. Performance of Random Projection (RP for short) and ℓ_1 filtering. (a) and (b) are the comparison under different rank ratios ($\rho_r \in [0.05, 0.15]$). (c) and (d) are the comparison under different data sizes ($m \in [1000, 5000]$). In (a) and (c), the y-axes are the CPU times (in seconds). In (b) and (d), the y-axes are the relative errors in log scale.

4.2. Structure from motion

In this subsection, we apply ℓ_1 filtering to a real world vision application, namely Structure from Motion (SfM). The problem of SfM is to automatically recover the 3D structure of an object from a sequence of images of the object. Suppose that the object is rigid, there are F frames and P tracked feature points (i.e., $\mathbf{L}_0 = [\mathbf{X}; \mathbf{Y}]_{2F\times P}$), and the camera intrinsic parameters do not change. As shown in [24], the trajectories of feature points from a single rigid motion of the camera all lie in a liner subspace of $\mathbb{R}^{2F}$, whose dimension is at most four (i.e., $\mathrm{rank}(\mathbf{L}_0) \le 4$). It has been shown that $\mathbf{L}_0$ can be factorized as $\mathbf{L}_0 = \mathbf{AB}$, where $\mathbf{A} \in \mathbb{R}^{2F\times 4}$ recovers the rotations and translations while the first three rows of $\mathbf{B} \in \mathbb{R}^{4\times P}$ encode the relative 3D positions for each feature point in the reconstructed object. However, when there exist errors (e.g., occlusion, missing data or outliers) the feature matrix is no longer of rank 4. Then recovering the full 3D structure of the object can be posed as a low-rank matrix recovery problem.

For this experiment, we first generate the 2D feature points $\mathbf{L}_0$ by applying an affine camera model (with rotation angles between 0 and 2π, with a step size $\pi/1000$, and uniformly randomly generated translations) to the 3D "Wolf" object,[12] which contains 4344 3D points. Then we add impulsive noises $\mathbf{S}_0$ (the locations of the nonzero entries are uniformly sampled at random) to part (e.g., 5% or 10%) of the feature points (see Fig. 6). In this way, we obtain corrupted observations $\mathbf{M} = \mathbf{L}_0 + \mathbf{S}_0$ with a size 4002×4344.

We apply our ℓ_1 filtering to remove outliers (i.e., $\mathbf{S}_0$) and compute the affine motion matrix $\mathbf{A}$ and the 3D coordinates $\mathbf{B}$ from the recovered features (i.e., $\mathbf{L}_0$). For comparison, we also include the results from the Robust Subspace Learning (RSL) [2],[13] and standard PCP (i.e., S-ADM based PCP, S-PCP for short). In Fig. 7, we show the original 3D object, SfM results based on noisy trajectories and trajectories recovered by RSL, S-PCP and ℓ_1 filtering, respectively. It is easy to see that the 3D reconstruction

[12] The 3D "Wolf" data is available at: http://tosca.cs.technion.ac.il/.

[13] The Matlab code of RSL is available at http://www.salleurl.edu/~ftorre/papers/rpca/rpca.zip and the parameters in this code are set to their default values.

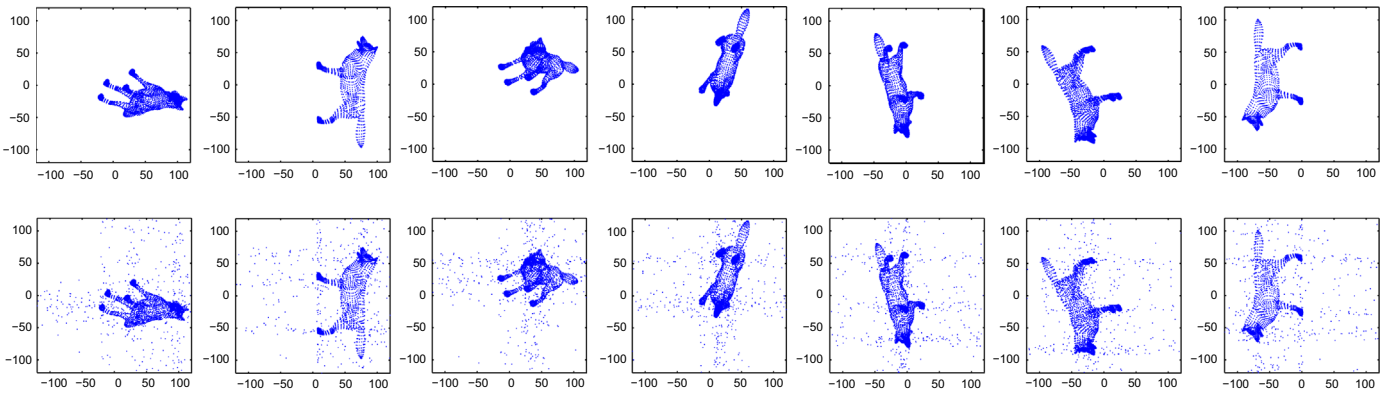

Fig. 6. The illustrations of some trajectories (2D image frames) generated by the 3D "Wolf" object (300th, 500th, …, 1300th, 1500th frames). Top row: the ground truth trajectories. Bottom row: 10% corrupted trajectories.

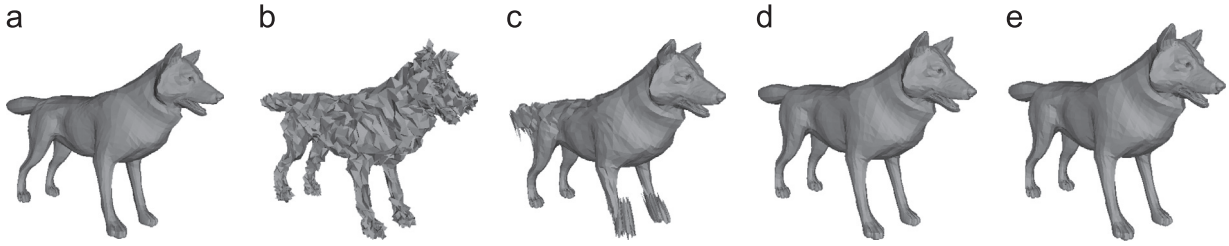

Fig. 7. The SfM reconstruction. (a) is the original 3D object. (b)–(e) are SfM results using corrupted trajectory and the trajectories recovered by RSL, S-PCP and ℓ_1 filtering, respectively. (a) Original, (b) Corrupted, (c) RSL, (d) S-PCP and (e) ℓ_1.

Table 2
Comparison among RSL, S-PCP and ℓ_1 filtering on the structure from motion problem. We present CPU time (in seconds) and the numerical accuracy of tested algorithms. $\mathbf{L}_0$ and $\mathbf{S}_0$ are the ground truth and $\mathbf{L}^*$ and $\mathbf{S}^*$ are the solution computed by different methods.

Noise	Method	RelErr	rank($\mathbf{L}^*$)	$\|\mathbf{S}^*\|_{\ell_0}$	Time	MaxDif($\mathbf{L}^*$)	AveDif($\mathbf{L}^*$)	ReprojErr
5%	rank($\mathbf{L}_0$) = 4, $\|\mathbf{S}_0\|_{\ell_0} = 869,234$							
	RSL	0.0323	4 (fixed)	15,384,229	93.05	32.1731	0.4777	0.9851
	S-PCP	5.18×10^{-9}	4	869,200	848.09	1.70×10^{-5}	2.47×10^{-8}	4.18×10^{-8}
	ℓ_1	1.16×10^{-8}	4	869,644	**6.46**	1.80×10^{-5}	3.61×10^{-7}	4.73×10^{-7}
10%	rank($\mathbf{L}_0$) = 4, $\|\mathbf{S}_0\|_{\ell_0} = 1,738,469$							
	RSL	0.0550	4 (fixed)	16,383,294	106.65	38.1621	0.9285	1.8979
	S-PCP	6.30×10^{-9}	4	1,738,410	991.40	1.57×10^{-5}	4.09×10^{-8}	6.82×10^{-7}
	ℓ_1	3.18×10^{-8}	4	1,739,912	**6.48**	5.61×10^{-5}	9.03×10^{-7}	1.26×10^{-6}

of RSL fails near the front legs and tail. In contrast, the S-PCP and ℓ_1 filtering provide results with almost the same quality. Table 2 further compares the numerical behaviors of these methods. We measure the quantitative performance for SfM by the well-known mean 2D reprojection error, which is denoted as "ReprojErr" and defined by the mean distance of the ground truth 2D feature points and their reprojections. We can see that the S-PCP provides the highest numerical accuracy while its time cost is extremely high (9 times slower than RSL and more than 100 times slower than ℓ_1 filtering). Although the speed of RSL is faster than S-PCP, its numerical accuracy is the worst among these methods. In comparison, our ℓ_1 filtering achieves almost the same numerical accuracy as S-PCP and is the fastest.

4.3. Background modeling

In this subsection, we consider the problem of background modeling from video surveillance. The background of a group of video surveillance frames is supposed to be exactly the same and the foreground on each frame is recognized as sparse errors. Thus this vision problem can be naturally formulated as recovering the low-rank matrix from its sum with sparse errors [11]. We compare our ℓ_1 filtering with other state-of-the-art robust approaches, such as RSL and S-PCP. For ℓ_1 filtering, we set the size of the seed matrix as 20×20.

For quantitative evaluation, we perform all the compared methods on the "laboratory" sequence from a public surveillance database [25] which has ground truth foreground. Both the False Negative Rate (FNR) and the False Positive Rate (FPR) are calculated in the sense of foreground detection. FNR indicates the ability of the method to correctly recover the foreground while the FPR represents the power of a method on distinguishing the background. These two scores correspond to the Type I and Type II errors in the statistical test theory[14] and are judged by the criterion that the smaller the better. One can see from Table 3 that RSL has the lowest FNR but the highest FPR among the compared methods. This reveals that RSL could not exactly distinguish the background. It can be seen that the performance of our ℓ_1 is as good as S-PCP, which achieves the best results but with the highest time cost.

To further test the performance of ℓ_1 filtering on large scale data set, we also collect a video sequence (named "Meeting") of 700 frames, each of which has a resolution 576×720. So the data

[14] Please refer to http://en.wikipedia.org/wiki/Type_I_and_type_II_errors.

matrix is of size greater than $700 \times 400{,}000$, which cannot be fit into the memory of our PC. As a result, we cannot use the standard ADM to solve the corresponding PCP problem. As for RSL, we have found that it did not converge on this data. Thus we only present the performance of ℓ_1 filtering. The time cost is reported in Table 3 and the qualitative results are shown in Fig. 8. We can see that the background and the foreground can be separated satisfactorily by ℓ_1 filtering. This makes sense because our ℓ_1 filtering can exactly recover the (global) low-rank structure for the background and remove the foreground as sparse errors. The speed of ℓ_1 filtering is also fast for such a large data set (with the dimensionality greater than 400,000).

Table 3
Comparison among RSL, S-PCP, and ℓ_1 filtering on background modeling problem. "Resolution" and "No. Frames" denote the size of each frame and the number of frames in a video sequence, respectively. We present FNR, FPR and the CPU time (in seconds) for the "Laboratory" data set. For our collected "Meeting" data set, we only report the CPU time because there is no ground truth foreground for this video sequence.

Video	Measure	RSL	S-PCP	ℓ_1
Laboratory	Resolution: 240×320, No. Frames: 887			
	FNR	7.31	8.61	8.62
	FPR	10.83	8.72	8.76
	Time	3159.92	10,897.96	**48.99**
Meeting	Resolution: 576×720, No. Frames: 700			
	Time	N.A.	N.A.	**178.74**

5. Extensions of PCP via ℓ_1 filtering

It has been shown in above sections that ℓ_1 filtering is a powerful tool for solving PCP model (2). In this section, we would like to further highlight some potential extensions of ℓ_1 filtering for more complex visual analysis problems. Here one should be aware that we do not claim that the strategy given in this section is the best solution to each problem. Instead, the goal is merely to show some examples of utilizing the mechanism of ℓ_1 filtering to handle different computer vision tasks.

5.1. Online subspace learning

Within the computer vision community, the dramatic increase in the amount of visual data has posed severe challenges for many problems, such as video processing, object tracking, motion analysis and human action segmentation and recognition. This is because the increasing volume of sequential data can overwhelm the traditional batch subspace learning approaches as they process all the test data simultaneously. To overcome this drawback, various online methods have been proposed in the computer vision society (e.g., [26,27]). But unfortunately, due to the difficulty in solving nuclear norm minimization incrementally at each iteration, it is hard to directly extend the existing PCP for online problems. In this subsection, we would like to discuss how to use the mechanism of ℓ_1 filtering to extend PCP for this challenge

Fig. 8. The sampled background modeling results of ℓ_1 filtering on the "Meeting" video sequence. (a) Is the original video sequence, (b)–(c) are the background ($\mathbf{L}^*$) and the foreground ($\mathbf{S}^*$) recovered by ℓ_1 filtering. (a) Original, (b) Background and (c) Foreground.

vision problem. First, the online subspace learning problem can be defined as follows:

Problem 5.1 (*Online subspace learning*). Let $[\mathbf{M}_1\ \mathbf{M}_2\ \cdots]$ be incrementally updated observations, in which $\mathbf{M}_1$ is the initially observed matrix and $\mathbf{M}_2, \ldots$ can only be achieved sequentially. Suppose the low-rank features of all submatrices (i.e., $\mathbf{M}_1, \mathbf{M}_2, \ldots$) are sampled from the same subspace. The goal is to calculate the following low-rank and sparse decomposition:

$$[\mathbf{M}_1\ \mathbf{M}_2\ \cdots] = [\mathbf{L}_1\ \mathbf{L}_2\ \cdots] + [\mathbf{S}_1\ \mathbf{S}_2\ \cdots]. \tag{20}$$

5.1.1. Online subspace learning via ℓ_1 filtering

We first discuss two naive strategies for Problem 5.1. First, it seems that solving (2) on each $\mathbf{M}_t$ ($t = 2, 3, \ldots$) can roughly address the above problem. However, when $\mathbf{L}_t$ is not low-rank or even full-rank relative to its size,[15] the model (2) can no longer correctly extract the intrinsic features for the data as the assumptions of PCP are violated. Another possible strategy is to solve (2) on a larger matrix $[\mathbf{M}_1\ \mathbf{M}_2\ \cdots\ \mathbf{M}_t]$ when we observed $\mathbf{M}_t$. However, this strategy has to suffer from extremely high computational cost as we have to recalculate (2) on all the data points when new test samples arrive.

Actually, by utilizing the mechanism of our proposed ℓ_1 filtering, we can efficiently address Problem 5.1 in the following way. First perform ℓ_1 filtering on $\mathbf{M}_1$ to recover $(\mathbf{L}_1, \mathbf{S}_1)$ and calculate the corresponding subspace basis $\mathbf{U}$.[16] Then $(\mathbf{L}_t, \mathbf{S}_t)$ can be easily computed by the solution of

$$\min_{\mathbf{S}_t, \mathbf{Q}_t} \|\mathbf{S}_t\|_{\ell_1} \quad \text{s.t. } \mathbf{M}_t = \mathbf{U}\mathbf{Q}_t + \mathbf{S}_t, \ \ t = 2, 3, \ldots, \tag{21}$$

and $\mathbf{L}_t = \mathbf{U}\mathbf{Q}_t$.

5.1.2. Applications to robust visual tracking

To show the power of our proposed model for real-world vision problems, we now apply the ℓ_1 filtering based online subspace learning strategy to address the visual tracking task. In particular, visual tracking can be considered as the problem of estimating the target object of next frame without knowing the concrete observation probability. Let $\mathbf{M}_t = [\mathbf{m}_1, \ldots, \mathbf{m}_t]$ denotes the observed images from the first frame to the tth frame and $\mathbf{x}_t$ denotes the state variable describing the affine motion parameters of an object at time t, respectively. Then we can process $\mathbf{x}_t$ with the following probabilities:

$$p(\mathbf{x}_t|\mathbf{M}_t) \propto p(\mathbf{m}_t|\mathbf{x}_t) \int p(\mathbf{x}_t|\mathbf{x}_{t-1}) p(\mathbf{x}_{t-1}|\mathbf{M}_{t-1})\, d\mathbf{x}_{t-1}, \tag{22}$$

where the dynamical model $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ denotes the state transition distribution and the observation model $p(\mathbf{m}_t|\mathbf{x}_t)$ estimates the likelihood of observing $\mathbf{m}_t$ at state $\mathbf{x}_t$. Following the work in [28], we set $\mathbf{x}_t$ as an affine transformation with six parameters and model the parameters by independent Gaussian distribution around the counterpart in $\mathbf{x}_{t-1}$, i.e., $p(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \mathbf{x}_{t-1}, \Psi)$, where Ψ denotes a diagonal covariance matrix. Then object tracking reduces to the problem of calculating the observation likelihood for sample state $\mathbf{x}_t$, i.e., $p(\mathbf{m}_t|\mathbf{x}_t)$. Let $\mathbf{m}_t^i$ be a candidate observation at newly coming frame, where i denotes the ith sample of the state $\mathbf{x}_t$. Then we can use low rank feature $\mathbf{l}_t^i$ and sparse error $\mathbf{s}_t^i$ to represent it, i.e., $\mathbf{m}_t^i = \mathbf{l}_t^i + \mathbf{s}_t^i$. In this way, the observation likelihood can be measured by the reconstruction error of each observed image patch,

$$p(\mathbf{m}_t^i|\mathbf{x}_t^i) = \exp(-\|\mathbf{m}_t^i - \mathbf{l}_t^i\|_2^2). \tag{23}$$

Based on the above preparation, the robust visual tracking problem can be successfully addressed by incrementally learning the low-rank features and the corresponding subspace basis for the target from corrupted observations using our extended ℓ_1 filtering method.

In the following, we compare the proposed tracking framework with six state-of-the-art visual tracking algorithms, i.e., Incremental Visual Tracking (IVT) [28], Multiple Instance Learning (MIL) [29], Visual Tracking by Sampling (VTS) [30], Tracking-Learning-Detection (TLD) [31], L1 Minimization (L1) [32,33] and Sparse Prototypes (SP) [34] on three challenge video sequences (i.e., "Football", "Singer" and "Cliffbar"[17]). As shown in Figs. 9 and 10 and Table 4, our tracking method (OUR for short) achieves the best performance in both qualitative and quantitative comparisons. The bottom row of Table 4 also compares the running time of three sparsity based methods (i.e., L1, SP and OUR). We observed that our ℓ_1 filtering based approach is the fastest one among them. This again verify the efficiency of our algorithm on real world vision problems.

5.2. Subspace clustering

Up to now, we all assume that the samples are drawn from a single subspace. However, a data set could be sampled from multiple subspaces in some vision problems, such as image segmentation, motion segmentation and face clustering. Therefore, it is necessary to simultaneously cluster the data into different subspaces and find a low-dimensional subspace to fit each group of points. As shown in [35], this so-called subspace clustering problem can be formulated as a Low-Rank Representation (LRR) model:

$$\min_{\mathbf{Z}, \mathbf{E}} \|\mathbf{Z}\|_* + \lambda\|\mathbf{E}\|_{\ell_1} \quad \text{s.t. } \mathbf{X} = \mathbf{X}\mathbf{Z} + \mathbf{E}, \tag{24}$$

where $\mathbf{X}$ is the observed data, $\mathbf{Z}$ is the low-rank representation and $\mathbf{E}$ is the sparse noise. The work in [36] has proved that LRR can exactly recover the multiple subspace structure. However, due to the SVD and the matrix–matrix multiplication at each iteration, the complexity of LRR is $O(n^3)$, thus hard to be applied to large-scale vision tasks. Even with accelerated techniques [22,37], it still suffers from an $O(n^2)$ computational complexity because model (24) involves more complex constraint than that in (2). Fortunately, as shown in [38], clean data $\mathbf{X} - \mathbf{E}$, rather than $\mathbf{X}$ itself, should be used as the dictionary. This observation reduces LRR to the PCP problem:

$$\min_{\mathbf{D}, \mathbf{E}} \|\mathbf{D}\|_* + \lambda\|\mathbf{E}\|_{\ell_1} \quad \text{s.t. } \mathbf{X} = \mathbf{D} + \mathbf{E}, \tag{25}$$

and then $\mathbf{Z} = \mathbf{V}_r\mathbf{V}_r^T$ [38], where $\mathbf{U}_r\Sigma_r\mathbf{V}_r^T$ is the skinny SVD of $\mathbf{D}$ and r is the rank of $\mathbf{D}$. In this view, by utilizing ℓ_1 filtering to handle (25), it is possible to provide a linear cost solver for LRR and hence for subspace clustering. Therefore, our ℓ_1 filtering is also a powerful tool for vision problems with data points drawn from multiple subspaces.

6. Conclusions and future work

In this paper, we propose the first *linear time* algorithm, named the ℓ_1 filtering, for *exactly* solving very large PCP problems, whose

[15] The matrix $[\mathbf{L}_1, \ldots, \mathbf{L}_t]$ is still low-rank relative to its size due to the assumption that all the features are drawn from the same subspace.

[16] When the size of the training data is relatively small, we can also simply perform S-PCP to extract the low-rank features $\mathbf{L}_1$ and then compute the subspace basis from its QR decomposition.

[17] The "Football" and "Singer" video sequences are available at http://cv.snu.ac.kr/research/~vtd/ and "Cliffbar" is available at http://vision.ucsd.edu/~bbabenko/project_miltrack.shtml.

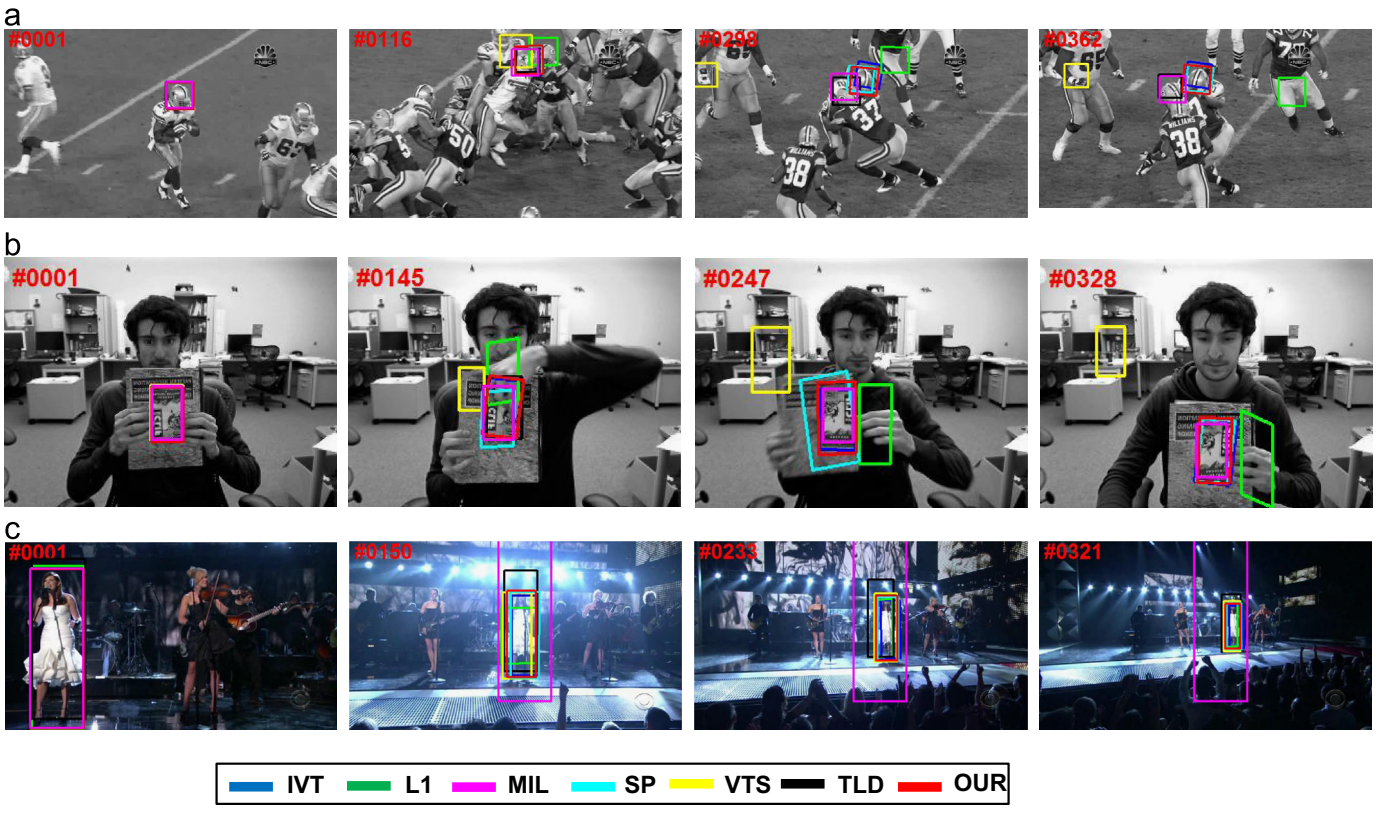

Fig. 9. Sampled tracking results on three challenging video sequences: (a) "Football", (b) "Cliffbar" and (c) "Singer".

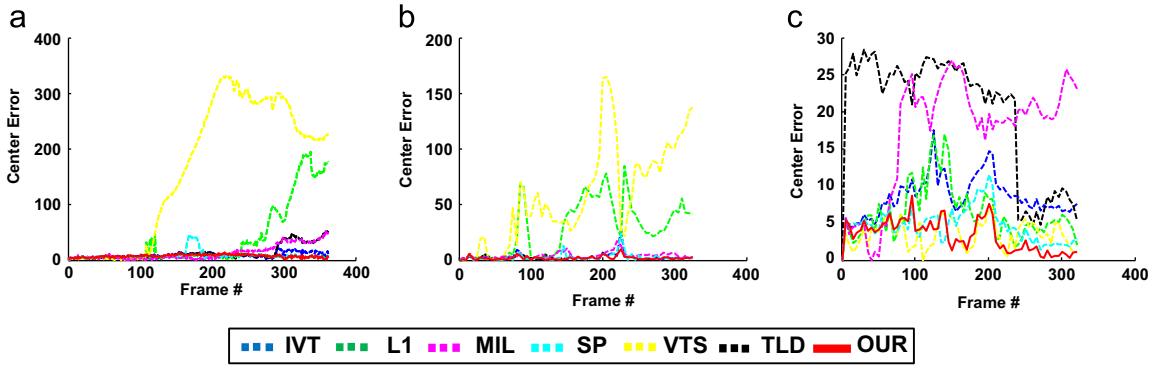

Fig. 10. Performance evolution using Center Location Error (CLE) in pixels on three challenging video sequences: (a) "Football", (b) "Cliffbar" and (c) "Singer".

Table 4
Quantitative comparisons using Center Location Error (CLE) in pixels. The bottom row also shows the average Frame per second (FPS) for our method and other two sparsity based trackers (i.e., L1 and SP) over all the frames. Here the FPS value of L1 is calculated using the APG-based acceleration technique [33], which is much faster than the original solver in [32].

Video	MIL	VTS	TLD	IVT	L1	SP	OUR
Football	13.4	161.8	13.5	9.4	39.9	8.9	**7.7**
Cliffbar	4.3	58.2	2.4	2.6	29.2	2.8	**2.0**
Singer1	17.2	3.5	19.9	8.5	6.4	4.7	**3.3**
Ave. CLE	11.6	74.2	11.9	6.8	25.2	5.5	**3.2**
Ave. FPS	–	–	–	–	1.0	1.7	**1.9**

ranks are supposed to be very small compared to the data size. It first recovers a seed matrix and then uses the seed matrix to filter some rows and columns of the data matrix. It avoids SVD on the original data matrix, and the ℓ_1 filtering step can be done in full parallelism. As a result, the time cost of our ℓ_1 filtering method is only linear with respect to the data size, making applications of RPCA to extremely large scale problems possible. The experiments on both synthetic and real world data demonstrate the high accuracy and efficiency of our method. However, there still remain several problems for further works.

- First, we have used a random sampling based technique to setup the ℓ_1 filtering algorithm. The effectiveness of such strategy has been verified by extensive experiments on both synthetic and real world problems. But it is still interesting to see whether such a Monte Carlo type method has an exact probabilistic guarantee for the problem.
- Also, the exact recoverability of ℓ_1 filtering is only guaranteed for data with large scale but relatively small intrinsic rank. This is because the standard PCP model is used to obtain the seed

matrix, which also needs a low-rank assumption. In Section 5.1, we have demonstrated that the column subspace basis can also be used for the ℓ_1 filtering process. However, such extension is still rudimentary in its current formulation. It is interesting to explore deeper theoretical analysis on the extended column subspace based ℓ_1 filtering scheme, especially for their recoverability on corrupted data set.

- Third, and more importantly, the mechanism of ℓ_1 filtering can be very useful for other large-scale computer vision problems. So the investigation on how to improve and augment ℓ_1 filtering for more vision applications, such as object detection and adaptive event detection, should also be considered.

Acknowledgments

The authors would like to thank Prof. Zaiwen Wen and Dr. Yadong Mu for sharing us their codes for LMaFit [19] and Random Projection [20], respectively. Risheng Liu is supported by the National Natural Science Foundation of China (No. 61300086), the China Postdoctoral Science Foundation, the Fundamental Research Funds for the Central Universities (No. DUT12RC(3)67) and the Open Project Program of the State Key Laboratory of CAD&CG, Zhejiang University, Zhejiang, China (No. A1404). Zhouchen Lin is supported by National Natural Science Foundation of China (Nos. 61272341, 61231002, 61121002). Zhixun Su is supported by National Natural Science Foundation of China (No. 61173103) and National Science and Technology Major Project (No. 2013ZX04005021). Junbin Gao is supported by Australian Research Council's Discovery Projects (No. DP130100364).

References

[1] J. Wright, A. Ganesh, S. Rao, Y. Peng, Y. Ma, Robust principal component analysis: exact recovery of corrupted low-rank matrices via convex optimization, in: NIPS, 2009, pp. 2080–2088.
[2] F. De la Torre, M. Black, A framework for robust subspace learning, Int. J. Comput. Vis. 54 (1–3) (2003) 117–142.
[3] Y. Peng, A. Ganesh, J. Wright, W. Xu, Y. Ma, RASL: robust alignment by sparse and low-rank decomposition for linearly correlated images, in: CVPR, 2010, pp. 763–770.
[4] L. Wu, A. Ganesh, B. Shi, Y. Matsushita, Y. Wang, Y. Ma, Robust photometric stereo via low-rank matrix completion and recovery, in: ACCV, 2010, pp. 703–717.
[5] Z. Zhang, A. Ganesh, X. Liang, Y. Ma, TILT: transform-invariant low-rank textures, Int. J. Comput. Vis. 99 (1) (2012) 1–24.
[6] F. Nie, H. Huang, C. Ding, D. Luo, H. Wang, Robust principal component analysis with non-greedy l_1-norm maximization, in: IJCAI, 2011, pp. 1433–1438.
[7] Q. Ke, T. Kanade, Robust l_1-norm factorization in the presence of outliers and missing data by alternative convex programming, in: CVPR, 2005, pp. 739–746.
[8] D. Skocaj, A. Leonardis, H. Bischof, Weighted and robust learning of subspace representations, Pattern Recognit. 40 (5) (2007) 1556–1569.
[9] Y. Deng, Q. Dai, R. Liu, Z. Zhang, S. Hu, Low-rank structure learning via nonconvex heuristic recovery, IEEE Trans. Neural Netw. Learn. Syst. 24 (3) (2013) 383–396.
[10] R. Liu, Z. Lin, F. De la Torre, Z. Su, Fixed-rank representation for unsupervised visual learning, in: CVPR, 2012, pp. 598–605.
[11] E. Candés, X. Li, Y. Ma, J. Wright, Robust principal component analysis? J. ACM 58 (3) (2011) 11.
[12] J. Wang, Y. Dong, X. Tong, Z. Lin, B. Guo, Kernel Nystrom method for light transport, ACM Trans. Graph. 28 (3) (2009) 29.
[13] A. Ganesh, Z. Lin, J. Wright, L. Wu, M. Chen, Y. Ma, Fast algorithms for recovering a corrupted low-rank matrix, in: Proceedings of International Workshop on Computational Advances in Multi-Sensor Adaptive Processing, 2009.
[14] Z. Lin, M. Chen, L. Wu, Y. Ma, The augmented Lagrange multiplier method for exact recovery of corrupted low-rank matrices, UIUC Technical Report UILU-ENG-09-2215.
[15] J. Cai, E. Candés, Z. Shen, A singular value thresholding algorithm for matrix completion, SIAM J. Optim. 20 (4) (2010) 1956–1982.
[16] R. Larsen, Lanczos Bidiagonalization with Partial Reorthogonalization, Department of Computer Science, Aarhus University, Technical Report, DAIMI PB-357.
[17] P. Drineas, R. Kannan, M. Mahoney, Fast Monte Carlo algorithms for matrices II: computing a low rank approximation to a matrix, SIAM J. Comput. 36 (1) (2006) 158–183.
[18] N. Halko, P. Martinsson, J. Tropp, Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions, SIAM Rev. 53 (2) (2011) 217–288.
[19] Y. Shen, Z. Wen, Y. Zhang, Augmented Lagrangian alternating direction method for matrix separation based on low-rank factorization, Optim. Meth. Softw. 29 (2) (2014) 239–263.
[20] Y. Mu, J. Dong, X. Yuan, S. Yan, Accelerated low-rank visual recovery by random projection, in: CVPR, 2011, pp. 2609–2616.
[21] X. Yuan, J. Yang, Sparse and low-rank matrix decomposition via alternating direction methods, Pacific J. Optim. 9 (1) (2014) 167–180.
[22] Z. Lin, R. Liu, Z. Su, Linearized alternating direction method with adaptive penalty for low rank representation, in: NIPS, 2011.
[23] E. Candés, M. Wakin, An introduction to compressive sampling, IEEE Signal Process. Mag. 25 (2) (2007) 21–30.
[24] S. Rao, R. Tron, R. Vidal, Y. Ma, Motion segmentation in the presence of outlying, incomplete, and corrupted trajectories, IEEE Trans. PAMI 32 (10) (2010) 1832–1845.
[25] C. Benedek, T. Szirányi, Bayesian foreground and shadow detection in uncertain frame rate surveillance videos, IEEE Trans. Image Process. 17 (4) (2008) 608–621.
[26] A. Jepson, D. Fleet, T. El-Maraghi, Robust online appearance models for visual tracking, IEEE Trans. PAMI 25 (10) (2003) 1296–1311.
[27] A. Levy, M. Lindenbaum, Sequential Karhunen–Loeve basis extraction and its application to images, IEEE Trans. Image Process. 9 (8) (2000) 1371–1374.
[28] D.A. Ross, J. Lim, R.-S. Lin, M.-H. Yang, Incremental learning for robust visual tracking, Int. J. Comput. Vis. 77 (1–3) (2008) 125–141.
[29] B. Babenko, M.-H. Yang, S. Belongie, Visual tracking with online multiple instance learning, in: CVPR, 2009.
[30] J. Kwon, K.M. Lee, Tracking by sampling trackers, in: ICCV, 2011.
[31] Z. Kalal, K. Mikolajczyk, J. Matas, Tracking-learning-detection, IEEE Trans. PAMI 34 (2012) 1409–1422.
[32] X. Mei, H. Ling, Robust visual tracking and vehicle classification via sparse representation, IEEE Trans. PAMI 33 (2011) 2259–2272.
[33] C. Bao, Y. Wu, H. Ling, H. Ji, Real time robust l_1 tracker using accelerated proximal gradient approach, in: CVPR, 2012.
[34] D. Wang, H. Lu, M.-H. Yang, Online object tracking with sparse prototypes, IEEE Trans. Image Process. 22 (2013) 314–325.
[35] G. Liu, Z. Lin, Y. Yu, Robust subspace segmentation by low-rank representation, in: ICML, 2010.
[36] G. Liu, Z. Lin, S. Yan, J. Sun, Y. Ma, Robust recovery of subspace structures by low-rank representation, IEEE Trans. PAMI. 35 (1) (2014) 171–184.
[37] R. Liu, Z. Lin, Z. Su, Linearized alternating direction method with parallel splitting and adaptive penalty for separable convex programs in machine learning, in: ACML, 2013.
[38] S. Wei, Z. Lin, Analysis and improvement of low rank representation for subspace segmentation, MSR-TR-2010-177.

Risheng Liu received the B.Sc and Ph.D degrees both in Mathematics from Dalian University of Technology in 2007 and 2012. He was a visiting scholar in Robotic Institute of Carnegie Mellon University from 2010 to 2012. He is currently a postdoctoral researcher in Faculty of Electronic Information and Electrical Engineering, Dalian University of Technology. His research interests include machine learning, pattern recognition, computer vision and numerical optimization.

Zhouchen Lin received the Ph.D. degree in applied mathematics from Peking University in 2000. He was a lead researcher in Visual Computing Group, Microsoft Research, Asia from 2000 to 2012. He is currently a Professor in Key Laboratory of Machine Perception (MOE), School of EECS, Peking University. His research interests include Image Processing, Pattern Recognition, Machine Learning, and Optimization. He is a senior member of the IEEE.

Zhixun Su received the B.Sc degree in Mathematics from Jilin University in 1987 and M.Sc degree in Computer Science from Nankai University in 1990. He received his Ph.D degree in 1993 from Dalian University of Technology, where he has been a professor in the School of Mathematical Sciences since 1999. His research interests include computer graphics and image processing, computational geometry, computer vision, etc.

Junbin Gao is currently a Professor in Computer Science in the School of Computing and Mathematics at Charles Sturt University (CSU). His recent research has involved the development of new machine learning algorithms for image analysis and computer vision.