Repairing Sparse Low-Rank Texture

Xiao Liang, Xiang Ren, Zhengdong Zhang, and Yi Ma, *Senior Member, IEEE,*

***Abstract*—In this paper, we show how to harness both low-rank and sparse structures in regular or near regular textures for image completion. Our method leverages the new convex optimization for low-rank and sparse signal recovery and can automatically correctly repair the global structure of a corrupted texture, even without precise information about the regions to be completed. Through extensive simulations, we show our method can complete and repair textures corrupted by errors with both random and contiguous supports better than existing low-rank matrix recovery methods. Through experimental comparisons with existing image completion systems (such as Photoshop) our method demonstrate significant advantage over local patch based texture synthesis techniques in dealing with large corruption, non-uniform texture, and large perspective deformation.**

***Index Terms*—Low-Rank and Sparse Matrix Recovery, Texture Completion, Image Repairing**

I. Introduction

Image completion has been an important but challenging problem widely studied in computer vision, computer graphics, and image processing in recent years. Given an image, with intensity values of certain regions missing (due to occlusion or corruption), the goal is to automatically recover or regenerate the missing pixel values in a way that the resulting image looks visually acceptable. This is inherently an extremely ill-conditioned problem as the statement of the problem does not ensure there will be a well-defined unique solution – in theory, one can fill in arbitrary values for the missing entries.

To make this problem more well-defined, people typically seek a solution such that the completed image is in some sense statistically or structurally consistent. More specifically, it requires that the pixel values of the missing regions follow the same statistical or geometric structures as the rest of the image. In the literature, based on different assumptions on the (local or global) statistics or structures of the input image, many methods which can directly or indirectly deal with image completion problems have been developed for different types of images or textures: from the synthesis of stationary stochastic random textures [1]–[3], to the inpainting of piece-wise smooth natural images [4]–[6] , and to the synthesis of highly symmetric and highly-structured regular textures [7].

In this paper, we focus on the class of images or textures whose structures have *very low intrinsic dimensionality or complexity*. More specifically we consider textures that when viewed as samples or signals in a high-dimensional space, span a very low-dimensional subspace or have a very sparse representation (w.r.t. certain basis). We call such textures as "Sparse Low-Rank Textures." A direct motivation for considering such a texture model is that many regular, symmetric patterns commonly seen in man-made environments (e.g. building facades and indoor decorations) are naturally sparse and low-rank. Nevertheless, this simple texture model actually encompasses a much richer class of textures. As we will see, our method works equally well when the structure to be completed is *not* strictly periodic or stochastically stationary. In fact, our method works even when the regular patterns are distorted by certain deformations such as a nonuniform scaling or a perspective transformation – which is typically the case for completing real images. See Figure 1 for a real example.

Another differentiating factor for image completion methods is their assumptions about the support of the missing pixels/regions, or equivalently, what types of missing regions they intend to handle. Are the missing pixels distributed uniformly or clustered as blocks in the image, how large can the missing regions be or do we want to extend the texture indefinitely? The last case is often referred to as texture synthesis, which we do not consider in this paper. Actually, almost all image completion and inpainting methods need to know the exact support of the missing pixels, and many are very sensitive to whether the support is given precisely (see Figure 3 for an example). This requirement has severely limited the applicability of image completion methods in practical scenarios. Very often we do not know the exact pixels in an image that need to be repaired. It is desirable that a method should be able to automatically identify and repair some corrupted or occluded regions whose statistics or structures are not consistent with the rest (say tree branches or a street sign blocking a building facade). When the support of the corrupted regions is only partially known or entirely unknown, we refer to the task as *image repairing*, to distinguish from the conventional image completion tasks with known support.

Contributions of This Paper. In this paper, we leverage recent breakthroughs in convex optimization for recovering sparse and low-rank structures and develop effective and efficient methods that can automatically repair sparse low-rank textures despite unknown or partially known corruptions and despite deformations (see Figure 1 for an example). We will show that by simultaneously enforcing the low-rank and sparse priors on a recovered image, we are able to handle much higher level of corruptions than conventional low-rank recovery techniques. We will show

Xiao Liang is with the Institute for Advanced Study, Tsinghua University, China, e-mail: liangx04@mails.tsinghua.edu.cn.

X. Ren is with Department of Computer Science, University of Illinois at Urbana Champaign.

Z. Zhang is with Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology.

Yi Ma is with Visual Computing Group, Microsoft Research Asia.

Manuscript received; revised.

Fig. 1. **A sample result produced by our method.** Left column: input image, where the green window denotes the input window to our system. Middle column: estimated support of corruption. Right column: repaired image.

with extensive simulations and experiments that for the class of sparse low-rank textures, our method achieves superior performance over existing image inpainting or image completion methods, even when the support of the corruption is provided for these methods. In particular, we compare thoroughly how well these methods can handle random errors, random block errors, large contiguous errors, or deformations. In the end, we show with numerous challenging examples how our method is readily applicable to many realistic image completion and repairing tasks.

Relations to Previous Work. There are many types of methods developed for image inpainting, completion or texture synthesis.

The first class of methods use generative parametric models (say stochastic PDEs) to describe the structures of the image signals. The completion processes essentially relies on the generalizability of such models and extrapolate the missing part of the image from the given one. For example, Bertalmio *et al.* ([8]) fills holes in an image by propagating image Laplacians in the isophote direction continuously from the exterior. Their PDE-based method has it roots in the Navier-Stokes equation for fluid dynamics ([9]). Oliveira *et al.* ([10]) proposed an image-based algorithm based on an isotropic diffusion model extended with the notion of user-defined diffusion barriers. Levin *et al.* ([11]) also performed image inpainting in the gradient domain using an image-specified prior.

Recently, there have also been works (see [12]–[14] and references therein) which perform inpainting based on sparse structures of image patches: patches inside the missing pixel region are synthesized as a sparse linear combination of elements from a patch dictionary. Furthermore, Bugeau *et al.* ([15]) combine geometric partial differential equation (PDEs) and patch-based texture synthesis in a variational model and performs image inpainting by minimizing the proposed energy function.

The above two classes of inpainting techniques work very well for natural images corrupted in small regions or thin gaps. However, for large missing regions that this paper will consider for image completion or repairing, they tend to generate blurring artifacts or often are not even applicable. Thus in our paper we will not make direct comparison with them.

In order to complete a large missing region or even extend a texture indefinitely, one must relies on more restrictive assumptions that the statistics or structures of the textures are stationary (for random textures) or homogenous (for regular patterns). The third class of methods rely on such an assumption and they essentially borrow and stitch together similar pixels or patches from given areas to complete the image. Such methods are widely used in computer graphics ([1]–[3], [5], [16], [17]). While these methods are designed to work for random textures, Liu *et al.* ([7]) shows how the patch-based techniques can be extended to work for regular or near regular symmetric patterns by explicitly estimating the underlying symmetric structures of the texture. In this work, we also work on regular or near regular textures. However, we will take a holistic approach that does not use any local statistics or patches. In addition, we directly impose regularity on the entire texture in terms of low-dimensionality and sparseness and hence completely bypass the difficult symmetry-estimation step.

Based on these studies, especially patch-based techniques, there are several works done specifically for image completion. Criminisi *et al.* ([6]) employs an exemplar-based texture synthesis technique modulated by a unified scheme for determining the fill order of the target region. Sun *et al.* ([5]) utilize user interaction to specify the structure information in the image and help to do image completion. Komodakis *et al.* ([4]) treats image completion, texture synthesis and image inpainting in a unified manner by posing all of the above image-editing tasks in the form of a discrete global optimization problem. Many effective image completion systems are based such patch-based techniques, including Patch Match (PM) that used by Photoshop ([18]), Image Completion with Structure Propagation (SP) developed by Microsoft ([5]), and Shift Map (SM) ([19]).

II. Low-rank Textures Inpainting

In this section, we describe the mathematical model for the class of *sparse low-rank textures* considered in this paper.

A. Definition of Low-rank Textures

We consider a 2D texture as a function $I(x, y)$, defined on $\mathbb{R}^2$. We say that I is a *low-rank texture* if the family of one-dimensional functions $\{I(x, y_0) \mid y_0 \in \mathbb{R}\}$ span a finite low-dimensional linear subspace *i.e.*,

$$r \doteq \dim\big(\text{span}\{I(x, y_0) \mid y_0 \in \mathbb{R}\}\big) \leq k \qquad (1)$$

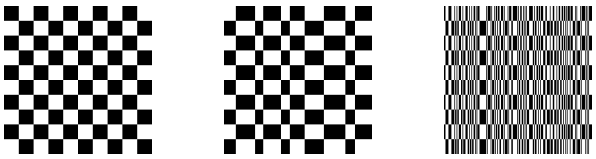


Fig. 2. **An example of rank's insufficiency.** These three images have exactly the same rank, but they go from purely regular, to nearly regular, and to nearly random textures.

for some small positive integer k. If r is finite, then we refer to I as a rank-r texture. To a large extent, the notion of low-rank texture unifies many of the conventional local features. By this definition, it is easy to see that *images of regular symmetric patterns always lead to low-rank textures*.

In practice, we are never given the 2D texture as a continuous function in $\mathbb{R}^2$. Typically, we only have its values sampled on a finite discrete grid in $\mathbb{Z}^2$, of size $m \times n$ say. In this case, the 2D texture $I(x, y)$ is represented by an $m \times n$ real matrix. For a low-rank texture, we always assume that the size of the sampling grid is significantly larger than the intrinsic rank of the texture *i.e.,*[1]

$$r \ll \min\{m, n\}.$$

It is easy to show that as long as the sampling rate is not one of the aliasing frequencies of the function I, the resulting matrix has the same rank as the continuous function[2]. Thus, the 2D texture $I(x, y)$ when discretized as a matrix, also denoted by I for convenience, has very low rank relative to its dimensions.

B. Recovery of Low-rank Textures

Suppose $D \in \mathbb{R}^{m \times n}$ is the given image and Ω is the set of observed entries (pixels). The goal is is to recover an image I of the lowest possible rank that agrees with D on all the observed pixels. This completion problem is equivalent to following problem:

$$\min_I \ \operatorname{rank}(I), \quad \text{s.t.} \ \ I_{ij} = D_{ij}, \forall (i, j) \in \Omega, \tag{2}$$

Although this is in general an NP-hard problem, in a recent seminal paper ([20]), Candès *et al.* has proved that under very mild conditions, even when a significant portion of entries are missing, the optimal low-rank solution can be found by solving the convex surrogate to the above problem:

$$\min_I \|I\|_*, \quad \text{s.t.} \ \ P_\Omega(I) = P_\Omega(D), \tag{3}$$

where $\|\cdot\|_*$ is the nuclear norm of a matrix (i.e. the sum of singular values) and P_Ω is a linear operator that restricts the equality only on the entries belong to Ω.

III. Sparse Low-rank Texture

A. Exploring Spatial Structure in Low-rank Texture

Although being low-rank is a necessary condition for most regular, structured images, it is certainly *not sufficient*. Figure 2 shows three images that have exactly the same rank. Obviously the first two are more smooth, realistic textures than the third one. As discussed in Section II-A, a rank-r texture is a 2D function $I(x, y)$, defined on $\mathbb{R}^2$. Then $I(x, y)$ can be factored as

$$I(x, y) = \sum_{i=1}^{r} g_i(x) h_i(y).$$

If I represents a more realistic regular or near regular pattern, it is typically piecewise smooth. Hence, the functions g_i and h_i are not arbitrary and they have additional structures. Piecewise smooth functions are typically sparse in certain transformed domain (say by Fourier or wavelet transform).

So, if we factorize the matrix I as

$$I = UV^T,$$

where U and V can be represented as

$$U = B_1 X_1 \ \text{ and } \ V = B_2 X_2$$

for some bases (B_1, B_2). If the bases are properly chosen, both X_1 and X_2 will be sparse. Or equivalently, the matrix $W = X_1 X_2^T$ will be a sparse matrix, which has the same (low) rank as I since $I = B_1 W B_2^T$.

B. Sparse Low-rank Texture Inpainting

Hence, if we want the recovered image to be both low-rank and sparse (in certain transformed domain), we could modify the above low-rank matrix completion problem (2) as follow to impose additional spatial structures:

$$\begin{array}{c} \min_{I,W} \operatorname{rank}(I) + \lambda \|W\|_0 \\ \text{s.t.} \ \ P_\Omega(I) = P_\Omega(D), \ I = B_1 W B_2^T, \end{array} \tag{4}$$

where $\|W\|_0$ denotes the number of non-zero entries in W. That is, we aim to find the texture I of the lowest possible rank and the transform matrix W with the fewest possible non-zero entries that agrees with the partial observation $P_\Omega(D)$ and the transformation $I = B_1 W B_2^T$. Here, λ is a weighting parameter which trades off the rank and sparsity of the recovered image.

In the above problem (4), both the rank function and the ℓ_0-norm are extremely difficult to optimization (in general NP-hard). Recent breakthroughs in sparse representation and low-rank matrix recovery have shown that under fairly broad conditions, they can be replaced by their convex surrogates ([20], [21], [22]): the matrix nuclear norm $\|I\|_*$ for $\operatorname{rank}(I)$ and the ℓ_1-norm $\|W\|_1$ for $\|W\|_0$, respectively. Thus, we end up with the following optimization problem:

$$\begin{array}{c} \min_{I,W} \|I\|_* + \lambda \|W\|_1 \\ \text{s.t.} \ \ P_\Omega(I) = P_\Omega(D), \ I = B_1 W B_2^T, \end{array} \tag{5}$$

If we further assume that the bases used are orthonormal, we have $\|I\|_* = \|B_1 W B_2^T\|_* = \|W\|_*$. The convex program (5) is equivalent to:

$$\min_W \quad \|W\|_* + \lambda \|W\|_1 \ \text{ s.t. } \ P_\Omega(B_1 W B_2^T) = P_\Omega(D). \tag{6}$$

[1] Notice that the scale of the window needs to be large enough to meet this assumption.

[2] In other words, the resolution of the image cannot be too low.

Algorithm 1 (Sparse Low-rank Texture Repairing)

INPUT: Input image $D \in \mathbb{R}^{m\times n}$, initial error support Ω^0, transformation bases B_1, B_2 and weights $\lambda > 0$, $\alpha > 0$.

WHILE not converged **DO**

Step 1 (inner loop): Solve the convex optimization:

$$(W^{i+1}, E^{i+1}) \leftarrow \operatorname{argmin}_{W,E} \|W\|_* + \lambda\|W\|_1 + \alpha\|E\|_1 \quad \text{s.t.} \quad P_{\Omega^i}(B_1 W B_2^T + E) = P_{\Omega^i}(D);$$

Step 2: Update the error support:

$$\Omega^{c,i+1} \leftarrow \Omega^{c,i} \cup \operatorname{supp}(E^{i+1}), \quad i \leftarrow i+1;$$

END WHILE

OUTPUT: Optimal solution W^*, E^* and repaired image $I^* = B_1 W^* B_2^T$.

IV. Sparse Low-rank Texture Repairing

Almost all previous methods for image inpainting or completion (e.g., [8], [12], [13], etc.) need information about the support Ω^c of the corrupted regions. This information is usually obtained through manually marked out by the user or detected by other independent methods. This often severely limits the applicability of all the image completion or inpainting methods.

In many practical scenarios, the information about the support of the corrupted regions might not be known or only partially known. Hence the pixels in the given region Ω can also contain some corruptions that violate the low-rank and sparse structures. We could model such corruptions as a sparse error term E, and solve the following optimization:

$$\begin{array}{ll} \min_{W,E} & \|W\|_* + \lambda\|W\|_1 + \alpha\|E\|_1 \\ \text{s.t.} & P_\Omega(B_1 W B_2^T + E) = P_\Omega(D). \end{array} \tag{7}$$

where α is a weighting parameter between sparsity and low-rankness. Notice that if we know nothing about the corruption areas, we only need to set Ω to be the entire image. Just like Robust PCA [21], the above convex program will decompose the image into a low-rank component and a sparse one. Of course, the nonzero entries in estimated E can help us to further refine the error support Ω^c. For instance, we could estimate the support by hard-thresholding the error:

$$\operatorname{supp}(E) = \{(i,j) \mid \forall |E_{ij}| > \varepsilon\} \tag{8}$$

for some threshold $\varepsilon > 0$. Or we could estimate the support of E using more sophisticated model to encourage additional structures such as spatial continuity. We will discuss one such option in Section VI.

We could further iterate between the image completion and support estimation till convergence. The repaired image is obtained as $I^* = B_1 W^* B_2^T$. The algorithm has been summarized as Algorithm 1. In practice, we also notice a good side effect of adding the additional E term, we can not only obtain the support of the corrupted regions but also can reduce noise on the repaired texture image I^*.

V. Solving Inner Loop via Linearized Alternating Direction Method

The most computationally expensive part of Algorithm 1 is solving the convex program (7) in the inner loop. This can be solved using conventional algorithms such as interior-point methods. While interior-point methods have excellent convergence properties, they do not scale very well with problem size and hence, unsuitable for real applications involving large images. There are recently a flurry of work in developing fast, scalable algorithms for nuclear norm minimization ([23]–[26]). To solve the convex optimization problem in (7), we use the linearized Augmented Lagrange Multiplier (LALM) method ([27], [28]). For the sake of completeness, in this section, we explain how the LALM method can be adapted to solve our problem.

A. Sketch of Linearized ADM

We first review the LALM algorithm in a more general setting, rather than for our specific problem. Let us consider convex optimization problems of the form:

$$\min_X f(X) \quad \text{s.t.} \quad \mathcal{A}(X) = b, \tag{9}$$

where f is a convex (not necessarily smooth) function, $\mathcal{A}$ is a linear function, and b is a vector of appropriate dimension. The basic idea of Lagrangian methods is to convert the above constrained optimization problem into an unconstrained problem that has the same optimal solution.

For the above problem (9), we can define the augmented Lagrange function as follows:

$$\mathcal{L}_\mu(X,Y) = f(X) + \langle Y, b - \mathcal{A}(X)\rangle + \frac{\mu}{2}\|b - \mathcal{A}(X)\|_F^2, \tag{10}$$

where Y is a Lagrange multiplier vector of appropriate dimension, $\|\cdot\|_F$ denotes the Frobenius norm, $\langle\cdot,\cdot\rangle$ denotes the matrix inner product, and $\mu > 0$ denotes the penalty imposed upon infeasible points. Theoretical result from ([27]) shows the important relationship between problem (9) and the augmented Lagrangian function (10).

ALM methods are a class of algorithms that simultaneously minimize the augmented Lagrangian function and compute an appropriate Lagrange multiplier. The classical ALM iteration proposed in [27] is given by

$$X_{k+1} = \arg\min_X \mathcal{L}_{\mu_k}(X, Y_k), \tag{11}$$

$$Y_{k+1} = Y_k + \mu_k\,(b - \mathcal{A}(X_k))\,, \tag{12}$$

$$\mu_{k+1} = \rho \cdot \mu_k, \tag{13}$$

where $\{\mu_k\}$ is a monotonically increasing positive sequence ($\rho > 1$). Thus, we have reduced the original optimization problem (9) to a sequence of unconstrained convex programs. Let $b_k := b + Y_k/\mu_k$, for (11) we have

$$X_{k+1} = \arg\min_X f(X) + \frac{\mu_k}{2}\|\mathcal{A}(X) - b_k\|_F^2 \tag{14}$$

The above iteration is computationally useful only if $\mathcal{L}_\mu(X,Y)$ is easy to minimize with respect to X. For the problem in our case, we need to approximated the subproblem of X by linearizing the quadratic term of its

objective function. With this linearization, the resulting approximation to (11) is then simple enough to have closed-form solution. More specifically, we have

$$\begin{aligned}\tfrac{1}{2}\|\mathcal{A}(X)-b_k\| \quad &\approx \quad \tfrac{1}{2}\|\mathcal{A}(X_k)-b_k\|+\langle g_k, X-X_k\rangle \\ &\quad +\tfrac{1}{2\eta}\|X-X_k\|_F^2\end{aligned} \tag{15}$$

where $\eta>0$ is a proximal parameter, and

$$g_k := \mathcal{A}^*(\mathcal{A}(X_k)-b_k) = \mathcal{A}^*(\mathcal{A}(X_k)-b-Y_k/\mu) \tag{16}$$

is the gradient of $\frac{1}{2}\|\mathcal{A}(X)-b_k\|_F^2$ at X_k. Plugging (16) into (14) and with simple manipulations, we obtain the following approximation to (14):

$$\min_X f(X)+\frac{\mu}{2\eta}\|X-(X_k-\eta g_k)\|_F^2. \tag{17}$$

This can be attributed to the following key property of the matrix nuclear norm and ℓ-1 norm:

$$\begin{aligned}\boldsymbol{T}_{\frac{\mu}{\eta}}(M) &= \operatorname{argmin}_X \|X\|_1+\tfrac{\mu}{2\eta}\|X-M\|_F^2, \\ \boldsymbol{S}_{\frac{\mu}{\eta}}(W) &= \operatorname{argmin}_X \|X\|_*+\tfrac{\mu}{2\eta}\|X-W\|_F^2.\end{aligned} \tag{18}$$

We use $\boldsymbol{S}_\varepsilon(\cdot)$ as the singular value shrinkage operator:

$$\boldsymbol{S}_\varepsilon(W) = U\boldsymbol{T}_\varepsilon(\Sigma)V^T$$

in which $U\Sigma V^T$ is the SVD of W and $\boldsymbol{T}(\cdot)$ represents the soft-thresholding operator which is defined on scalars as follows:

$$\boldsymbol{T}_\varepsilon(x) = \operatorname{sgn}(x)\cdot(|x|-\varepsilon).$$

The soft-thresholding operator is extended to vectors and matrices by applying it elementwise.

B. Solving Inner Loop via LADM

To solve the convex program (7), we utilize the Linearized Alternating Direction Method (LADM) given by Lin *et al.*([26]). This method has proven to be one of the fastest algorithms for solving various low-rank matrix completion and recovery problems. To adopt LADM method to our problem, we need to make our objective function separable. Thus we introduce an auxiliary variable A to replace W in the low-rank term, which turns the optimization problem into the following:

$$\begin{aligned}&\min_{A,E,W} \|A\|_*+\lambda\|W\|_1+\alpha\|E\|_1 \\ &\text{s.t.}\quad A=W,\ P_\Omega(B_1WB_2^T+E)=P_\Omega(D).\end{aligned} \tag{19}$$

For the problem (19), the augmented Lagrangian function is defined as:

$$\begin{aligned}&\mathcal{L}_\mu(A,W,E,Y_1,Y_2) = \|A\|_*+\lambda\|W\|_1+\alpha\|E\|_1 \\ &+\langle Y_1, A-W\rangle+\langle Y_2, P_\Omega(B_1WB_2^T+E)-P_\Omega(D)\rangle \\ &+\tfrac{\mu}{2}\|A-W\|_F^2+\tfrac{\mu}{2}\|P_\Omega(B_1WB_2^T+E)-P_\Omega(D)\|_F^2,\end{aligned} \tag{20}$$

where Y_1, Y_2 are the Lagrange multipliers, $\mu>0$ is a penalty parameter.

From the previous discussion in Section V-A, the classic linearized ALM iteration scheme for our problem is given by

$$\begin{aligned}&(A_{k+1},W_{k+1},E_{k+1}) = \operatorname{argmin} L_{\mu_k}(A,W,E,Y_{1,k},Y_{2,k}), \\ &Y_{1,k+1} = Y_{1,k}+\mu_k\cdot P_\Omega(B_1W_{k+1}B_2^T+E_{k+1}-D), \\ &Y_{2,k+1} = Y_{2,k}+\mu_k\cdot(A_{k+1}-W_{k+1}), \\ &\mu_{k+1} = \rho\cdot\mu_k,\end{aligned} \tag{21}$$

where $\rho>1$ is a constant.

We now focus on efficiently solving the first step of the above iterative scheme. In general, it is computationally expensive to minimize over all the variables A, W and E simultaneously. So, we adopt a common strategy to solve it *approximately* by adopting an *alternating minimizing* strategy *i.e.,* minimizing with respect to A, W and E one at a time:

$$A_{k+1} = \operatorname*{argmin}_A \mathcal{L}_{\mu_k}(A,W_k,E_k,Y_{1,k},Y_{2,k}), \tag{22}$$

$$W_{k+1} = \operatorname*{argmin}_W \mathcal{L}_{\mu_k}(A_{k+1},W,E_k,Y_{1,k},Y_{2,k}), \tag{23}$$

$$E_{k+1} = \operatorname*{argmin}_E \mathcal{L}_{\mu_k}(A_{k+1},W_{k+1},E,Y_{1,k},Y_{2,k}) \tag{24}$$

Subproblem (22) is easy to optimize and we can get to the closed-form solution without doing any approximation:

$$\begin{aligned}A_{k+1} &= \operatorname*{argmin}_A \|A\|_*+\frac{\mu_k}{2}\big\|A-W_k+\frac{1}{\mu_k}Y_{1k}\big\|_F^2 \\ &= \boldsymbol{S}_{\frac{1}{\mu_k}}\big(W_k-\frac{1}{\mu_k}Y_{1k}\big).\end{aligned} \tag{25}$$

Subproblems (23) and (24) need linear approximation since there are linear operator on variables E and W. As we discuss in Section V-A, we can approximate the objective function by linearizing the quadratic penalty term in the augmented Lagrangian function (20) and adding a proximal term to update W and E. This results in the following updating scheme:

$$\begin{aligned}W_{k+1} &= \operatorname*{argmin}_W \lambda\|W\|_1+\frac{\mu_k\eta_1}{2}\|W-M_k\|_F^2 \\ &= \boldsymbol{T}_{\frac{\lambda}{\mu_k\eta_1}}(M_k),\end{aligned} \tag{26}$$

and

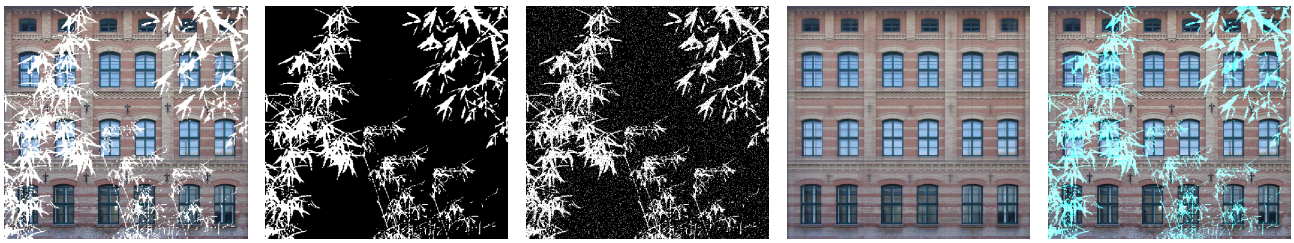
$$\begin{aligned}E_{k+1} &= \operatorname*{argmin}_E \alpha\|E\|_1+\frac{\mu_k\eta_2}{2}\|E-N_k\|_F^2 \\ &= \boldsymbol{T}_{\frac{\alpha}{\mu_k\eta_2}}(N_k),\end{aligned} \tag{27}$$

where

$$\begin{aligned}&M_k = W_k-\Big(B_1^TO_kB_2+W_k-A_{k+1}-Y_{1,k}/\mu_k\Big)/\eta_1, \\ &N_k = E_k-O_k/\eta_2 \\ &O_k = P_\Omega(E_k+B_1W_{k+1}B_2^T-D)+Y_{2,k}/\mu_k\end{aligned}$$

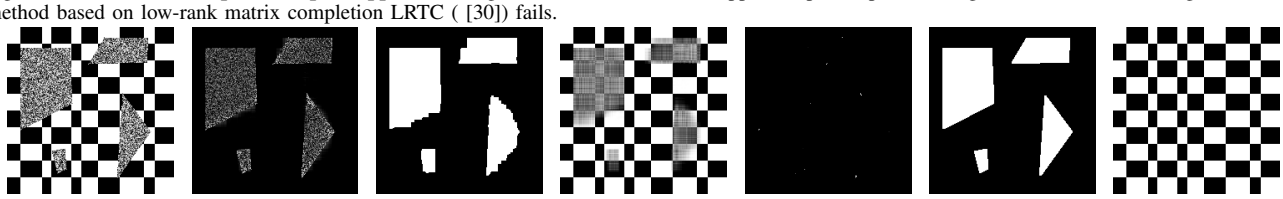
Here, $\eta_1>0$ and $\eta_2>0$ are some parameters (cf. [29]), and we set $\eta_1=3$, $\eta_2=3$ in our paper. We summarize the ADM scheme for solving (7) as Algorithm 2. We choose the sequence $\{\mu_k\}$ to satisfy $\mu_{k+1}=\rho\,\mu_k$ for some $\rho>1$.

To show the advantage of adding the robust error term E for support detection, we do a comparison with another low-rank texture completion method based on matrix completion ([30]). In Figure 3, we first corrupted an input low-rank image with a ground truth support Ω, then we generate a new support $\widetilde{\Omega}$ by randomly switching 5% of the support Ω (from 1 to 0 or from 0 to 1). We use $\widetilde{\Omega}$ as the input support for our method and the LRTC method ([30]) and compare their completion results. In the comparison, we set parameters $\lambda=0.001$, $\alpha=0.85$ for our algorithm and use DCT as the basis for B_1 and B_2. For the LRTC method, all parameters are set as the same values as those in the



(a) Input image (b) True supp. (c) Input supp. (d) Our result (e) LRTC result

Fig. 3. **Robustness to imperfect input support.** This figure shows that with 5% support map corrupted, our algorithm recovers the image while the method based on low-rank matrix completion LRTC ([30]) fails.



(a) Input D (b) E^1 (c) Ω^1 (d) I^1 (e) E^2 (f) Ω^2 (g) I^2

Fig. 4. **Estimation of contiguous error support using MRF.** This figure shows an example of estimating a contiguous support by our method after merging MRF into our model. It takes only two iterations for our method to converge precisely to the correct support.

Algorithm 2 (Solving Inner Loop via LADM)

INPUT: Input image $D \in \mathbb{R}^{m\times n}$, error support Ω, transformation bases B_1, B_2, and weights $\lambda > 0$, $\alpha > 0$.
Initialization: $k = 0, A_0 = D, E_0 = 0, W_0 = 0, , Y_{1,0} = 0, Y_{2,0} = 0, \mu_0 > 0, \rho > 1$;
WHILE not converged **DO**
Update A_{k+1} using Eq. (25);
Update W_{k+1} using Eq. (26);
Update E_{k+1} using Eq. (27);
$Y_{1,k+1} = Y_{1,k} + \mu_k \cdot P_\Omega(B_1 W_{k+1} B_2^T + E_{k+1} - D)$;
$Y_{2,k+1} = Y_{2,k} + \mu_k \cdot (A_{k+1} - W_{k+1})$;
$\mu_{k+1} = \rho\mu_k$;
END WHILE
OUTPUT: solution (A, W, E) to problem (7).

public code of LRTC. The results in Figure 3 clearly show that our method can handle erroneous input support while other matrix completion methods would fail.

VI. Handling Error E with Contiguous Support

In a real image, very often the regions we need to repair have contiguous supports – occluded or corrupted pixels are likely to be adjacent to each other in the image. The simple hard-thresholding used in (8) treats each pixel independently and does not take this additional information for estimating the error support. To incorporate such prior information, we could use a Markov random field (MRF) to model the spatial continuity of the error support $\mathrm{supp}(E)$, as in [31]. We do not make any other assumptions about the locations, sizes, shapes, or the number of occluded regions.

Following a similar strategy in [31] to estimate a continuity-prone support, we only have to replace the support update in equation (8) in Algorithm 1 by the following:

$$\begin{aligned}\mathrm{supp}(E) \;=\; & \mathrm{argmin}_{\Omega\in\{-1,1\}^{m\times n}} \textstyle\sum_{(l,n)\in M} \gamma s[i]s[j] \\ & + \textstyle\sum_{l\in V} \log\big(p(e^i[l]|s[l])\big),\end{aligned} \tag{28}$$

with

$$\begin{aligned}\log\big(p(e[l]|s[l]=-1)\big) &= \begin{cases} -\log\beta, & |e[l]| \le \beta; \\ \log\beta, & |e[l]| > \beta. \end{cases} \\ \log\big(p(e[l]|s[l]=1)\big) &= \begin{cases} 0, & |e[l]| > \beta; \\ \log\beta, & |e[l]| \le \beta. \end{cases}\end{aligned} \tag{29}$$

where $G = (V, M)$ is a graph which represents the image domain, $s = \mathrm{vec}(\Omega) \in \mathbb{R}^{mn}$, $e = \mathrm{vec}(E) \in \mathbb{R}^{mn}$, $V = \{1, ..., mn\}$ denotes the set of $m \times n$ pixels and M denotes the edges connecting neighboring pixels. Here, the parameter β indicate the level of error we would accept before considering an entry of the image as occluded. The parameter γ in the Markov random field model controls the strength of mutual interaction between adjacent pixels. It should correspond to the level of spatial continuity for the error supports.

Figure 4 shows how MRF works. We used $\beta = 0.015, \gamma = 5$ and $\lambda = 0.001, \alpha = 0.85$ in the example (empirically set since they bring best performance). All other experiments use the same parametric setting unless stated otherwise.

VII. Handling Deformed Texture D

In practice, textures in a real image that we need to perform the completion or repairing task rarely have precise regular patterns. Very often we see a distorted version of the regular patterns. For instance, a building facade seen through a perspective camera introduces a planar homography between the facade plane and the image plane.

In this case, the recovered low-rank texture I agrees with the observed texture D up to certain transformation. To recover I, we then need to solve the following problem instead:

$$\begin{aligned}&\min_{W,E,\tau} \|W\|_* + \lambda\|W\|_1 + \alpha\|E\|_1 \\ &\text{s.t.}\;\; P_\Omega(B_1 W B_2^T + E) = P_\Omega(D\circ\tau),\end{aligned} \tag{30}$$

where $\tau : \mathbb{R}^2 \to \mathbb{R}^2$ belongs to a certain group of transforms, e.g., affine transform, perspective transforms (see

Algorithm 3 (Deformed Sparse Low-rank Texture Repairing)

INPUT: Input image $D \in \mathbb{R}^{m\times n}$, initial transformation $\tau \in \mathbb{G}$(affine or projective), initial error support Ω^0, Jacobian matrix J, transformation bases B_1, B_2 and weights $\lambda > 0$, $\alpha > 0$.

WHILE not converged **DO**

Step 1 (inner loop): Solve the convex optimization (30):

$$\begin{aligned} &\min_{W,E,\Delta\tau} \|W\|_* + \lambda\|W\|_1 + \alpha\|E\|_1 \\ &\text{s.t.} \quad P_{\Omega^i}(B_1 W B_2^T + E) = P_{\Omega^i}(D\circ\tau^i + J\Delta\tau), \end{aligned}$$

Step 2: Update the transformation:

$$\tau^{i+1} = \tau^i + \Delta\tau^{i+1},$$

Step 3: Update the error support:

$$\Omega^{c,i+1} \leftarrow \Omega^{c,i} \cup \mathrm{supp}(E^{i+1}), \quad i \leftarrow i+1;$$

END WHILE

OUTPUT: Optimal solution W^*, E^*, τ^* and repaired image $I^* = B_1 W^* B_2^T$.

[32]), and even general cylindrical deformation ([33])(see Section VIII-D **A**).

Above problem is not a convex program as the constraint is now nonlinear in the unknowns. Similar to the work of TILT by [32], we could linearize $D\circ\tau$ at the previous τ^i as

$$D\circ(\tau^i + \Delta\tau) \approx D\circ\tau^i + J\Delta\tau,$$

where J is the Jacobian: derivative of the image with respect to the transformation parameters. Then, to handle deformation, our method can be easily modified from (30) as follows:

$$\begin{aligned} &\min_{W,E,\Delta\tau} \|W\|_* + \lambda\|W\|_1 + \alpha\|E\|_1 \\ &\text{s.t.} \quad P_{\Omega}(B_1 W B_2^T + E) = P_{\Omega}(D\circ\tau + J\Delta\tau), \end{aligned} \tag{31}$$

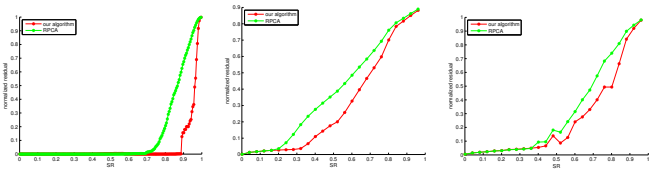
for ease of representation, we will omit the subscript i on Ω. We summarized the algorithm to handle deformation τ in the sparse low-rank texture repairing work as Algorithm 3.

Details of solving (30) is similar to solving (7) with linearized ADM method. We omit the detail of the inner loop and for readers who interested in it can refer to the appendix. The algorithm runs in the same spirit as TILT, except that it further imposes sparsity of the recovered texture and repair the image once the locations of errors are detected. Nevertheless the improvement of our method in repairing the rectified texture is very significant. Figure 5 shows an example that compares the results of our method with the TILT code released by the authors of [32].

VIII. Simulations and Experiments

A. Comparison with Low-rank Matrix Recovery

Here we conduct simulations to study the working range of our method. We corrupt different percentages of a clean texture (without deformation) with gross errors and see how much corruption we can handle for different types of corruption (random, one large block, random small blocks).



(a) random corruption (b) one disk (c) random blocks

Fig. 6. **Quantitative comparisons with RPCA.** This figure shows normalized residual error versus the sampling rate of our method and RPCA [21] on different kinds of corruptions.

To demonstrate the importance of the sparse prior on recovering natural low-rank textures and the effectiveness of the proposed solution, we compare our method with the state-of-art low-rank matrix completion or recovery method: Robust Principal Component Analysis (RPCA) [21] (The object function is $\min_A \|A\|_*$ s.t. $P_\Omega(A) = P_\Omega(D)$, which does not exploit the sparse prior at all). This is also the method which some of the latest texture completion methods such as LRTC [30] are based upon.

In this simulation, we restrict to the simpler case that the support Ω is precisely known (hence RPCA is equivalent to matrix completion). The ratio $p/(m\times n)$ between the number of corrupted/missing entries (pixels) and the number of entries in the matrix is denoted by "SR" (sampling ratio). We use the normalized residual $\|A-M\|_F/\|M\|_F$ to measure the recovery error, where A is the recovered image, and M is the ground truth.

The parameter setting for our algorithm and RPCA is $\lambda = 0.001, \alpha = 0$. We use DCT as basis in all simulations and experiments unless otherwise stated. We have tested three kind of corruptions: uniform random corruptions, one disk corruption, and random block corruptions on three representative low-rank textures: a synthesized checkerboard image and two real texture images. The checkerboard is of precise rank 2 and the other two are full rank but approximately low-rank.

We test the two algorithms for input images with SR growing from 0 to 99.9%. Figure 6 shows quantitative comparison between the two methods with different levels and kinds of corruptions. Figure 7 further show some qualitative results that compare the two methods. These results clearly show that the sparse prior on the image indeed helps low-rank texture completion in all cases.

B. Handling Error E with MRF

In this section, we shows that the MRF strategy (28) not only can handle contiguous support better than hard-threshold, but also works on sparse support after set the mutual interaction control $\gamma = 0$. According to equation (29), the minimization function would degenerate to hard threshold if we set $\gamma = 0$. We can see that the support estimate result by MRF with $\gamma = 0$ is exactly the same as hard-thresholding (8). So usually we use $\gamma = 5$ to handle continuous support, and set$\gamma = 0$ when the support is sparse such as tree branches.

(a) Input image (b) Our result (c) Our support (d) TILT result (e) E of TILT

Fig. 5. **Comparison with TILT.** Notice that TILT does not (intend to) fully repair the occluded parts while our method does.

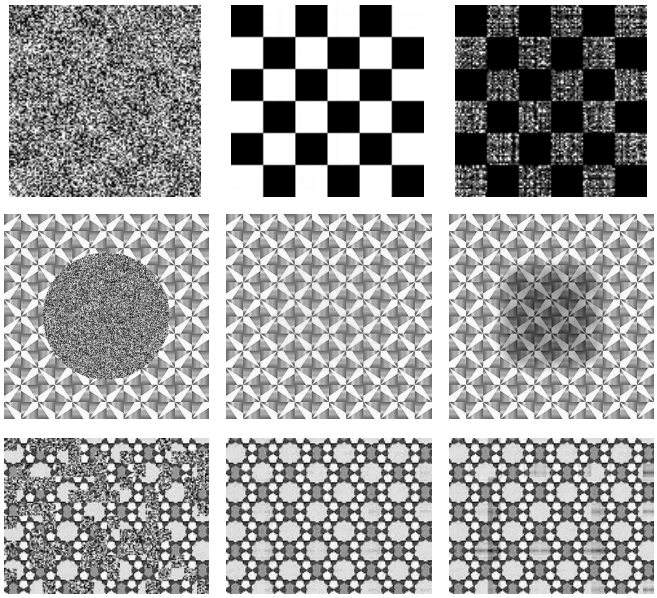
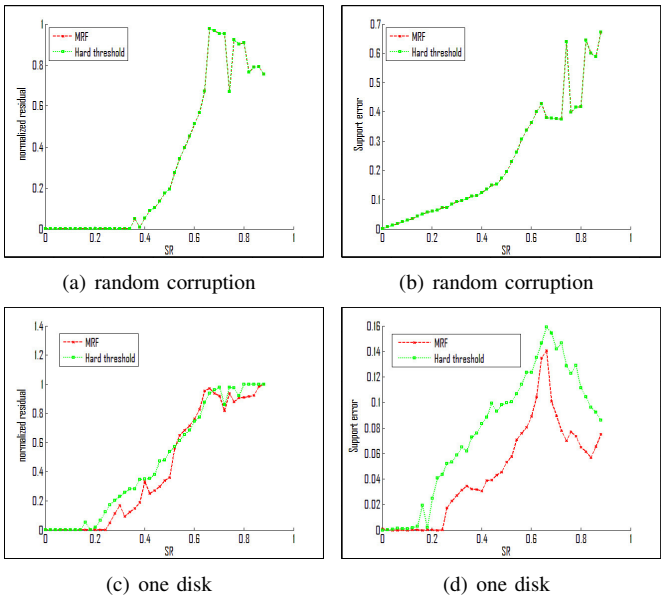


Fig. 7. **Qualitative comparisons with RPCA on simulated images.** First row is a checkerboard texture 91% randomly corrupted; second row is a real texture 30% corrupted on a disk like region; and third row is a real texture 40% corrupted by random blocks. In each case, the first image is input, the second is our result, and the third is result by RPCA [21].



(a) random corruption (b) random corruption

(c) one disk (d) one disk

Fig. 8. **Qualitative comparisons with hard-thresholding on a checkerboard image.** Images on the left column show normalized residual error of recovered image versus the sampling rate of MRF and hard-thresholding; Images on the right column show normalized error of estimated support versus the sampling rate of MRF and hard-thresholding.

This experiments compare MRF with hard threshold on two types of support. The parameter setting for MRF is $\gamma = 0, \beta = 0.3$ for random support and $\gamma = 5, \beta = 0.015$ for continuous support. For hard-thresholding, $\Omega^i = |E| > \beta$. The iteration will be terminated when $\|\Omega^{i+1} - \Omega^i\|_1 < mn/100$. The input error support $\Omega^0 \in \mathbb{R}^{m\times n}$ is set to be all zero. We can see from Figure 8 that MRF is better than hard-thresholding with contiguous support and has exactly the same result with hard-thresholding when $\gamma = 0$. If not specified, all the other experiments in this section use MRF to build support with the same parameters and strategy as here.

C. Comparison with Image Completion Systems

In this experiment, we have conducted a series of comparative tests between our method and a few recent image completion methods such as the Shift Map (SM) method [19] and some highly engineered commercial systems: Patch Match (PM) used by Photoshop ([18], [34]), Image Completion with Structure Propagation (SP) developed by Microsoft Research Asia [5]. These methods all share the spirit of sample-based texture synthesis: they stitch sampled local patches together to ensure certain global consistency. As these methods rely mostly on local statistics and structures, they tend to work on natural images or random textures too while our method does not. However, this experiment is to demonstrate significant advantages of our method on completing or repairing *regular or near regular* low-rank patterns. The reason is partially because these methods normally do not or cannot exploit global structural information about the textures.

Unlike our method, these image completion systems typically require the user to mark out rather precisely the to-be-completed region or regions (marked out as red regions in this paper), and even to provide additional information about the structures to be recovered (such as that required by the SP method). Notice that if the support of the corrupted region is small, our method does not even need information about the support at all. For our method, we only have to specify a large window that contains some corrupted regions (see Figure 9 forth row first column for example). Nevertheless, if the corruption is somewhat too large (say larger than 25% of the input region), our method can also take a rough support for the corrupted region Ω^0 as part of the input (marked with red curves in the input images shown in Figure 10). The repairing result can be

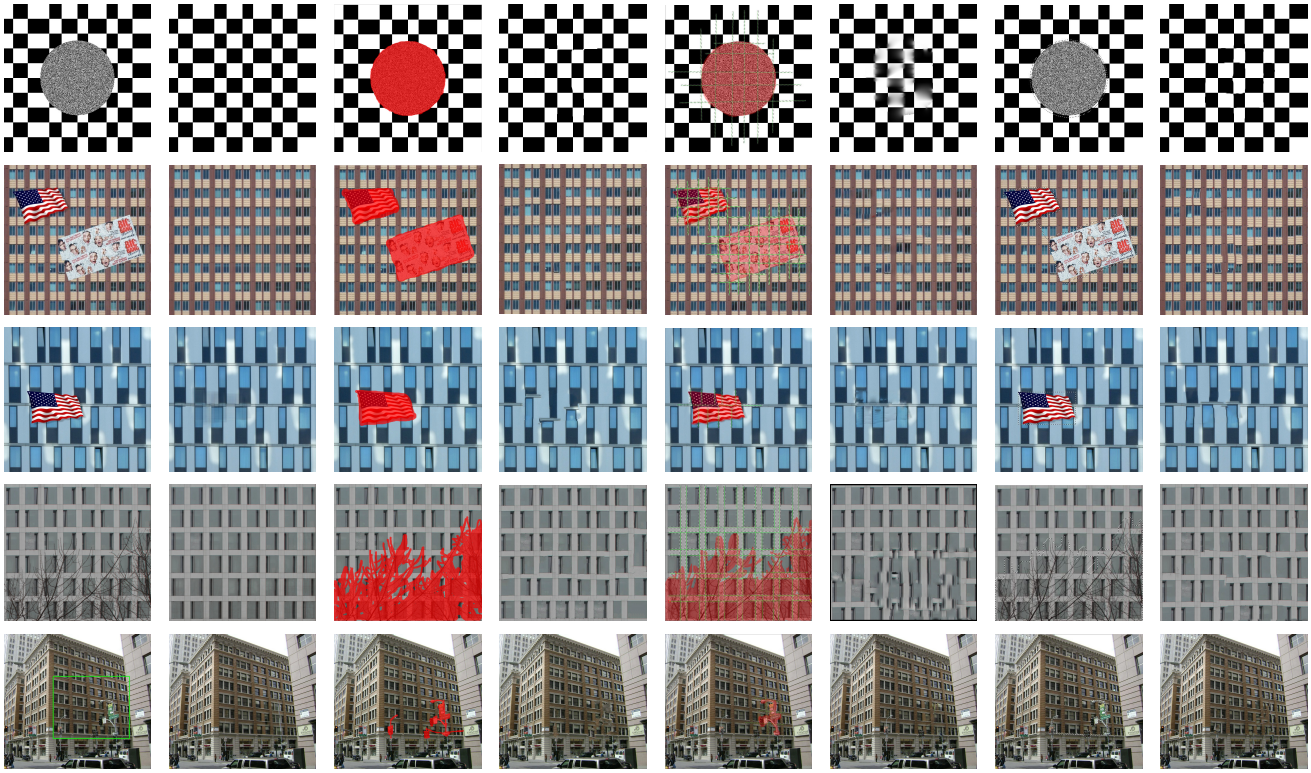


Fig. 9. **Comparison results with Shift-Map [19], SP [5] and Photoshop [18].** Columns 1-2: input and result of our method; Columns 3-4: input and result of Shift-Map; Columns 5-6: input and result of SP; Columns 7-8: input and result of Photoshop. For rectified textures, our method can take the whole image as input, while in the deformed case, a green window indicates our input.

significantly improved. As our method is inherently robust, the provided support needs not to be so precise.

In addition, these methods are not designed at all to handle textures that are deformed from a purely homogeneous texture or regular pattern. For instance, they can not handle images with perspective deformation. In our method, the deformation can be estimated by solving (31). The estimated τ is then used to rectify the input region and we then solve (19) to repair the texture, typically with two iterations (first to estimate the error support, second to complete the image with a partially known error support). If the image has no deformation, we can simply skip the deformation estimation step. For a color image, three matrices associated with the three RGB channels are repaired independently. In this experiment, we set $\alpha = 0.85$, and all other parameters remain the same as previous simulations and experiments.

In Figure 9, we present comparisons with Shift-Map method [19], Patch Match (PM) ([18], [34]) and Structure Propagation (SP) [5] on five different images: a simulated non-uniform low-rank texture, a uniform building facade, a somewhat less uniform building facade, a real building with tree branch occlusion and a real building facade image with perspective deformation, which correspond to the five rows in Figure 9, respectively. As we see, these three methods all have trouble in recovering correctly or accurately the structural patterns of the original images.

D. Real Examples of Image Repairing

In this section, we test the efficacy of our algorithm on natural images belonging to various categories. Besides some examples where our algorithm works very well, we also present some cases that are particularly challenging where our algorithm succeeds only to some extent, and some examples where it fails completely. We believe that from these examples, we may gain a better understanding of both the strengths and limitations of the our algorithm. To show how our method can handle realistic image repairing tasks, Figure 10 shows several interesting examples produced by our method on real urban scene images. The process to generate these results is exactly the same as that in previous comparison experiments. The parameters for the first image is $\lambda = 0.001, \alpha = 0.85, \gamma = 0, \beta = 0.3$, and $\lambda = 0.001, \alpha = 0.85, \gamma = 5, \beta = 0.015$ for all the other images.

Curved surface cases. The current algorithm is only a basic version and its capability is still limited, especially when we try to apply it to cases where the assumptions are not fully met. Through the remainder of this section and the next section, we will discuss some of the limitations, as well as potential extensions that make it work better in some of the more challenging cases. Figure 11 shows an extension to images with low rank texture on a generalized cylindrical surface. This extension generalizes our techniques from planar surfaces to a much larger class of important 3D surfaces. To handle such generalized cylindrical surfaces, we only need a more complicated parametric model for τ

Fig. 10. **Realistic repairing results.** First row: input images where the green windows denote the input windows to our system, and red contours denote the initial error supports to our system. Second row: repairing images by our method. Third row: estimated final error supports by our method.

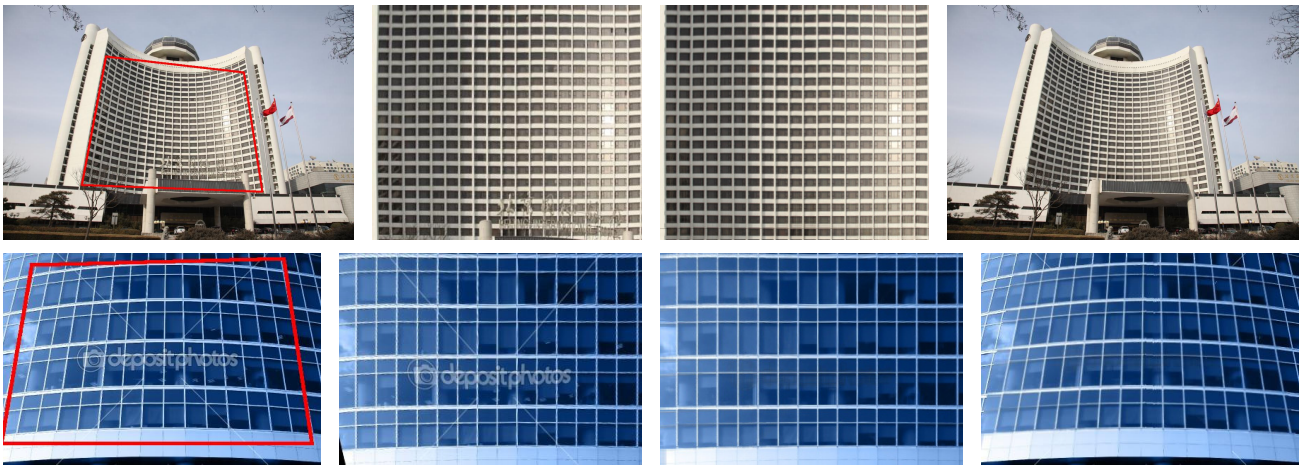


Fig. 11. **Curved low-rank texture.** Images on the left column are the input, windows with red border are the original input; Images in the second column are the rectified low-rank textures (after deformation undone by our algorithm); Images in the third column are the repaired low-rank textures; Images on the right are the repaired textures mapping back to the input images.

than plane cases, which parameterize the unknown camera pose and surface shape. For details, please see [33].

Challenges and limitations. As our algorithm is designed to be robust to sparse errors, sometimes it may tend to make correction to (sparse) boundaries if the error tolerance is set to be high. Figure 12 shows such cases. For the image with pink dots that we used in the experiment of Figure 10, we have changed the parameter β from 0.3 to 0.02. The result changes a lot compared to that in the first column of in Figure 10. The second row of Figure 12 shows a similar example. Sometimes, an input image contains multiple regions, each of which might be distorted differently. In this case, we need to deal with them one after another in order to obtain satisfactory results. Figure 13 shows such an example.

Expected failures. Although our algorithm works very robustly under very broad conditions and for a wide range of near regular textures, it may fail to hallucinate the missing part correctly if the conditions are too challenging or the assumptions are violated. Figure 14 shows some of the cases. The first example shows the limitations of the percentage of the support region in the input low rank texture. When the occluded region is too large, our algorithm tend to converge at an optimal solution with dark inpainting result. The repairing result is limited by the nuclear convex surrogate. This drawback would be overcome if we can handle low rank minimization better. The second example is a facade with not only occlusion but also severe reflections.

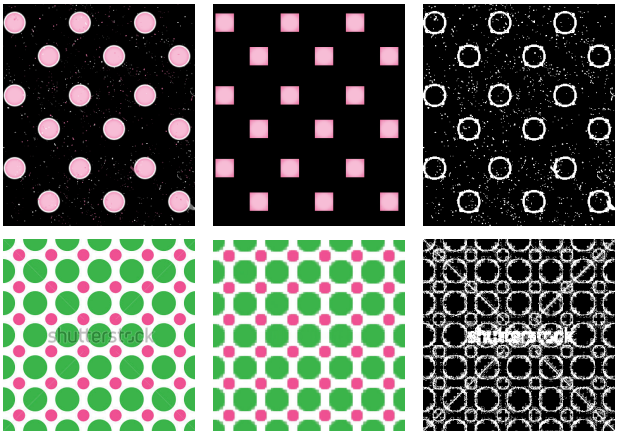


Fig. 12. **Over-correction cases.** From left to right : the input image, the output image, the estimated support. Our algorithm over-corrects around edges in the input image.

Fig. 13. **Two regions with different distortion.** The first image is the input image; the second image is the result after our method applied to the top-half; the third image is to apply our method to the bottom half; the last image is the final result.

The repairing result is reasonable but not perfect. Only low rank is not enough to handle these challenging cases. We need more information such as more input images to make a good recovery. As mentioned earlier in Section 2, our algorithm is not designed to work on random textures occurring in nature, such as the last one shown in Figure 14. Although there has been work in the literature showing that it is possible to repair a reasonable approximation of the lawn based on the statistical properties of the random texture, our algorithm is certainly not designed to handle such cases. It is effective for regular symmetric textures, but not for random textures which normally have high-rank as matrices.

IX. Potential Modifications and Extensions

As we see in the above results, our algorithm could recover the low-rank and sparse structures in regular or near regular textures very well. However, because the output of our algorithm is the optimization result of a convex problem, our output result is sometimes not as sharp as the original input image. To handle this problem, one may add a post-processing step based on a simple patch-matching method: simply borrow raw pixels from the input image based on local similarity. A naive process can be as follows: For every pixel in the repaired region, we take a, say 15×15, patch centered at the pixel, and find its closest matched patch in the original input image. We then replace the pixel value with that from the original image. After this post-processing step, all pixels of the final image

Fig. 14. **Some failure cases.** Our algorithm fails to work well on those images since they deviate significantly from the conditions under which the algorithm is designed to work. From left to right: too much occlusion; large occlusion and reflection; random (high-rank) textures.

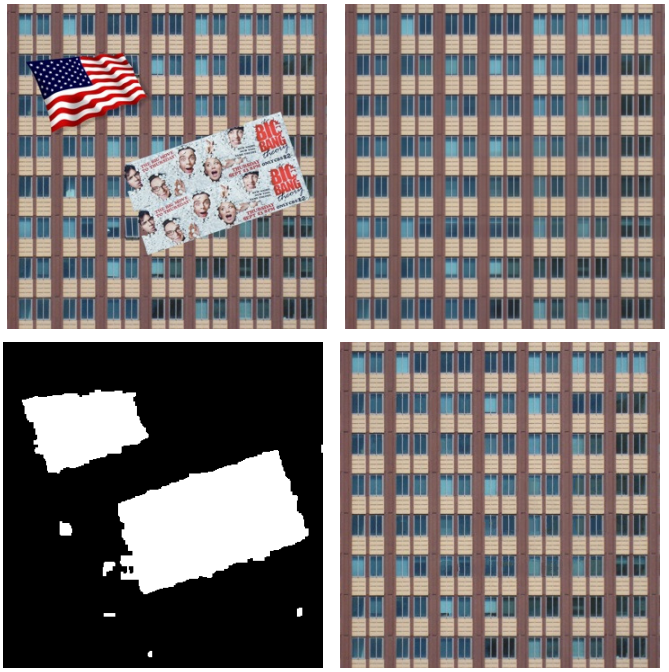


Fig. 15. **Patch-matching based post-processing.** Top row: the input image and the recovered image by our method; Bottom: estimated support and result after post-processing, respectively. (Subtle difference between the two results on the right side is better noticed in the electronic version).

now come from the original image. Both sharpness and noise-characteristics would be the same as the input image. Figure 15 shows such an example. This suggests that our method can be combined with existing local patch based texture synthesis methods. The result system not only could remedy the limitations of both approaches, it could even simultaneously handle both regular and random textures. We leave such work for future investigation.

References

[1] A. A. Efros and T. K. Leung, "Texture synthesis by non-parametric sampling," *IEEE International Conference on Computer Vision (ICCV)*, pp. 1033–1038, 1999.

[2] A. A. Efros and W. T. Freeman, "Image quilting for texture synthesis and transfer," *Proceedings of SIGGRAPH 2001*, pp. 341–346, August 2001.

[3] L. Liang, C. Liu, Y. Xu, B. Guo, and H. Shum, "Real-time texture synthesis by patch-based sampling," *ACM Transactions on Graphics (SIGGRAPH)*, vol. 20, pp. 127–150, 2001.

[4] N. Komodakis and G. Tziritas, "Image completion using global optimization," 2006.
[5] J. Sun, L. Yuan, J. Jia, and H.-Y. Shum, "Image completion with structure propagation," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 861–868, 2005.
[6] A. Criminisi, P. Pérez, and K. Toyama, "Object removal by exemplar-based inpainting," *Proc. IEEE Computer Vision and Pattern Recognition (CVPR)*, 2003.
[7] Y. Liu, W.-C. Lin, and J. Hays, "Near-regular texture analysis and manipulation," *ACM Transactions on Graphics (SIGGRAPH)*, vol. 23, no. 3, pp. 368–376, 2004.
[8] M. Bertalmio and G. Sapiro, "Image inpainting," *ACM Transactions on Graphics (SIGGRAPH)*, pp. 417–424, 2000.
[9] M. Bertalmio, A. L. Bertozzi, and G. Sapiro, "Navier-stokes, fluid dynamics, and image and video inpainting," *Proc. IEEE Computer Vision and Pattern Recognition (CVPR)*, pp. 355–362, 2001.
[10] M. M. Oliveira, B. Bowen, R. Mckenna, and Y. sung Chang, "Fast digital image inpainting," *Proceedings of the International Conference on Visualization, Imaging and Image Processing (VIIP)*, 2001.
[11] A. Levin, A. Zomet, and Y. Weiss, "Learning how to inpaint from global image statistics," *Proceedings of the Internetional Conference on Computer Vision (ICCV)*, 2003.
[12] J. Mairal, M. Elad, and G. Sapiro, "Sparse representation for color image restoration," *the IEEE Trans. on Image Processing (TIP)*, vol. 17, pp. 53–69, 2007.
[13] M. J. Fadili, J. l. Starck, and F. Murtagh, "Inpainting and zooming using sparse representations," *The Computer Journal*, p. 6479, 2009.
[14] M. Elad, J. l. Starck, P. Querre, and D. Donoho, "Simultaneous cartoon and texture image inpainting using morphological component analysis (MCA)," *The Computer Journal*, pp. 340–358, 2005.
[15] A. Nigeau, M. Bertalmio, V. Caselles, and G. Sapiro, "A comprehensive framework for image inpainting," *the IEEE Trans. on Image Processing (TIP)*, vol. 19, pp. 2634 – 2645, 2010.
[16] V. Kwatra, A. Schodl, I. Essa, G. Turk, and A. Bobick, "Graphcut textures: Image and video synthesis using graph cuts," *ACM Transactions on Graphics (SIGGRAPH)*, vol. 22, no. 3, pp. 277–286, July 2003.
[17] J. Hays and A. A. Efros, "Scene completion using millions of photographs," *ACM Transactions on Graphics (SIGGRAPH)*, vol. 26, no. 3, 2007.
[18] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman, "PatchMatch: A randomized correspondence algorithm for structural image editing," *ACM Transactions on Graphics (SIGGRAPH)*, vol. 28, no. 3, 2009.
[19] Y. Pritch, E. Kav-Venaki, and S. Peleg, "Shift-map image editing," *Proceedings of the Internetional Conference on Computer Vision (ICCV)*, 2009.
[20] E. J. Candès and B. Recht, "Exact matrix completion via convex optimization," *Foundations of Computational Mathematics*, vol. 9, no. 6, pp. 717–772, 2009.
[21] E. Candès, X. Li, Y. Ma, and J. Wright, "Robust principal component analysis?" *Journal of the ACM*, vol. 58, no. 7, May 2011.
[22] V. Chandrasekaran, S. Sanghavi, P. A. Parrilo, and A. S. Willsky, "Rank-sparsity incoherence for matrix decomposition," *SIAM Journal on Optimization*, vol. 21, no. 2, pp. 572–596, 2011.
[23] J. Cai, E. Candès, and Z. Shen, "A singular value thresholding algorithm for matrix completion," *SIAM Journal on Optimization*, vol. 20, no. 4, pp. 1956–1982, 2010.
[24] K. Toh and S. Yun, "An accelerated proximal gradient algorithm for nuclear norm regularized least squares problems," *Pacific Journal of Optimization*, vol. 6, pp. 615–640, 2010.
[25] A. Ganesh, Z. Lin, J. Wright, L. Wu, M. Chen, and Y. Ma, "Fast algorithms or recovering a corrupted low-rank matrix," *proceedings of Third International Workshop on Computational Advances in Multi-Sensor Adaptive Processing*, December 2009.
[26] Z. Lin, M. Chen, L. Wu, and Y. Ma, "The augmented Lagrange multiplier method for exact recovery of corrupted low-rank matrices," *UIUC Technical Report UILU-ENG-09-2215*, 2010.
[27] D. P. Bertsekas, *Nonlinear Programming*. Athena Scientific, 2004.
[28] J. Yang and X. Yuan, "Linearized augmented lagrangian and alternating direction methods for nuclear norm minimization," *Mathematics of Computation*, To appear.
[29] Z. Lin, R. Liu, and Z. Su, "Linearized alternating direction method with adaptive penalty for low-rank representation," *Advances in Neural Information Processing Systems (NIPS)*, 2011.
[30] J. Liu, P. Musialski, P. Wonka, and J. Ye, "Tensor completion for estimating missing values in visual data," pp. 2114–2121, 2009.
[31] Z. Zhou, A. Wagner, H. Mobahi, J. Wright, and Y. Ma, "Face recognition with contiguous occlusion using markov random fields," *in Proceedings of IEEE International Conference on Computer Vision (ICCV)*, 2009.
[32] Z. Zhang, A. Ganesh, X. Liang, and Y. Ma, "TILT: Transform-invariant low-rank textures," *International Journal of Computer Vision (IJCV)*, 2012.
[33] Z. Zhang, X. Liang, and Y. Ma, "Unwrapping low-rank textures on generalized cylindrical surfaces," *International Conference on Computer Vision (ICCV)*, 2011.
[34] C. Barnes, E. Shechtman, D. Goldman, and A. Finkelstein, "The generalized patchmatch correspondence algorithm," *European Conference on Computer Vision (ECCV)*, 2010.