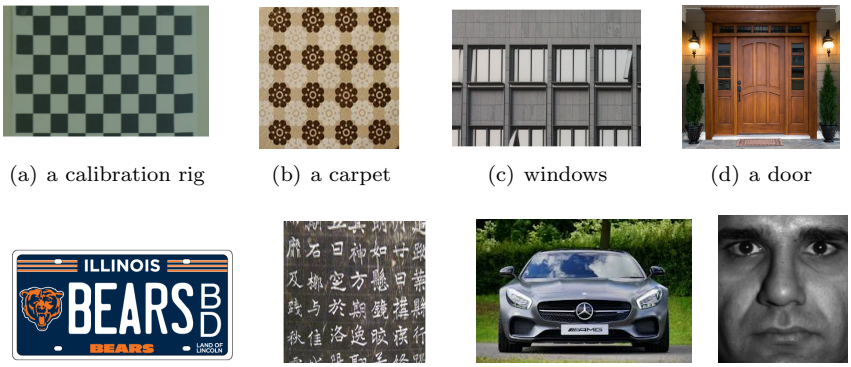
Chapter 14
Structured Texture Recovery

14.1 Introduction

In man-made environments, most objects of interest are rich of regular, repetitive, symmetric structures. Figure 14.1 shows images of some representative structured objects. An image of such an object clearly inherits such regular structures and encodes rich information about the 3D shape, pose, or identity of the object. If we view the image of such an object as a matrix, columns of the matrix will obviously be correlated to one another hence the rank of the matrix will be very low, or approximately so. For example, for reflectively symmetric objects like a face or a car, the rank of their images will be at most half of the size of the matrices. Besides being symmetric, images of such objects typically have other additional structures (e.g. piecewise smooth etc) which will render the rank of the image even much lower.[1] We generally refer to images (or image regions) associated with such structure objects as "structured textures" to separate them from other textures.

In this chapter, we will study how the low-dimensional structures of such structured textures may help us to robustly and accurately recover the appearance, pose, and shape of the associated objects in 3D. This makes structured textures extremely important for many computer vision tasks

[1] The reader should test validity of this assumption by computing the actual rank of real images similar to those in Figure 14.1. We leave this as an exercise.



(a) a calibration rig (b) a carpet (c) windows (d) a door

(e) a license plate (f) characters (g) a car (h) a face

Figure 14.1. Representative examples of structured objects.

such as recognition, localization, and reconstruction of objects in man-made environments. From a compressive sensing perspective, we will see in this chapter how to recover a low-rank matrix (that models structured textures) despite significant corruption (due to occlusion) or transformation[2] (due to pose, shape or camera lens distortion etc.) To be more precise, we first introduce some notation.

14.2 Low-rank Textures

Strictly speaking, an image (viewed as a matrix[3]) is a discrete sampling of a continuous texture (function) defined on a 2D domain. Consider a 2D texture as a function $\boldsymbol{I}^0(x, y)$, defined in $\mathbb{R}^2$. We say that $\boldsymbol{I}^0$ is a *low-rank texture* if the family of one-dimensional functions $\{\boldsymbol{I}^0(x, y_0) \mid y_0 \in \mathbb{R}\}$ span a finite low-dimensional linear subspace *i.e.*,

$$r \doteq \dim(\text{span}\{\boldsymbol{I}^0(x, y_0) \mid y_0 \in \mathbb{R}\}) \leq k \tag{14.2.1}$$

for some small positive integer k. If r is finite, then we refer to $\boldsymbol{I}^0$ as a rank-r texture. It is easy to see that a rank-1 function $\boldsymbol{I}^0(x, y)$ must be of the form $u(x) \cdot v(y)$ for some functions $u(x)$ and $v(y)$; and in general, a rank-r function $\boldsymbol{I}^0(x, y)$ can be explicitly factorized as as the combination of r rank-1 functions:

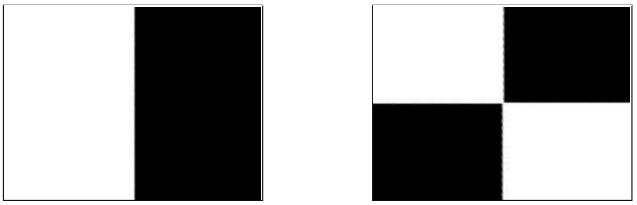
$$\boldsymbol{I}^0(x, y) \doteq \sum_{i=1}^{r} u_i(x) \cdot v_i(y). \tag{14.2.2}$$

[2] in the 2D domain of the matrix

[3] Hence, in this chapter, we will use the bold face symbol $\boldsymbol{I}$ to denote an image because we will mostly identify the image as a matrix.

Figure 14.2 shows some ideal low-rank textures: edges and corners were traditionally used in computer vision to characterize local features of an object and they can be viewed as the simplest low-rank textures. An ideal vertical edge (or slope) as shown in Figure 14.2 left can be considered a rank-1 texture with $u(x) = -\text{sign}(x)$ and $v(y) = 1$. An ideal corner as shown in Figure 14.2 is also a rank-1 texture with $u(x) = \text{sign}(x)$ and $v(y) = \text{sign}(y)$.

Thus, in a sense, the notion of low-rank textures unifies many of the conventional local features but goes beyond that: By its definition, it is easy to see that *images of regular, repetitive, symmetric patterns typically lead to low-rank textures*. Low-rank textures of rank higher than one would be able to represent much richer class of structure objects than local edges or corners and they can capture the global characteristics of an structure object.



(a) an edge (b) a corner

Figure 14.2. Examples of some ideal low-rank textures: an edge and a corner.

Given a low-rank texture, obviously its rank is *invariant* under any scaling of the function, as well as scaling or translation in the x and y coordinates. That is, if

$$\boldsymbol{I}(x, y) \doteq \alpha \cdot \boldsymbol{I}^0(ax + t_1, by + t_2)$$

for some constants $\alpha, a, b \in \mathbb{R}_+, t_1, t_2 \in \mathbb{R}$, then $\boldsymbol{I}(x, y)$ and $\boldsymbol{I}^0(x, y)$ have the same rank according to our definition in (14.2.1). For most practical purposes, it suffices to recover any scaled or translated version of the low-rank texture $\boldsymbol{I}^0(x, y)$, as the remaining ambiguity left in the scaling can often be easily resolved in practice by imposing additional constraints on the texture. Hence, in this paper, unless otherwise stated, we view two low-rank textures *equivalent* if they are scaled and translated versions of each other:

$$\boldsymbol{I}^0(x, y) \sim \boldsymbol{I}^0(ax + t_1, by + t_2),$$

for some $a, b, c \in \mathbb{R}_+, t_1, t_2 \in \mathbb{R}$. In homogeneous representation, this equivalence group of transformations consists of all elements of the form:

$$g \in \mathbb{G} \doteq \left\{ \begin{bmatrix} a & 0 & t_1 \\ 0 & b & t_2 \\ 0 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{3\times 3} \,\middle|\, a, b \in \mathbb{R}_+, t_1, t_2 \in \mathbb{R} \right\}. \tag{14.2.3}$$

It is however easy to see that the low-rank form 14.2.2 will *not* be preserved under a general linear transform of the domain:

$$\boldsymbol{I}(x, y) \doteq \boldsymbol{I}^0(ax + bx + t_1, cx + dy + t_2) \tag{14.2.4}$$

will have a different (usually much higher) rank than that of $\boldsymbol{I}^0(x, y)$. For instance, if we rotate the edge or the corner in Figure 14.2 by 45°, the resulting image (as a matrix) will become full rank. Similarly, the rank will mostly increase under more general nonlinear distortions or transformations of the domain. As we will see in this chapter, this fact is actually rather beneficial: it suggests that *the correct transformation that can undo the distortion would be the one that makes the rank of the texture the lowest.*

In practice, an image of a 2D texture is not a continuous function defined on $\mathbb{R}^2$. 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 $\boldsymbol{I}^0(x, y)$ is represented by an $m \times n$ matrix of real numbers. 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,[4] i.e.

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

It is easy to show that as long as the sampling rate is not one of the aliasing frequencies of the functions $u_i(x)$ or $v_i(x)$ of the continuous $\boldsymbol{I}^0(x, y)$ defined in (14.2.2), the resulting matrix has the same rank as the continuous function.[5] Thus, the 2D texture $\boldsymbol{I}^0(x, y)$ when discretized as a matrix, denoted by $\boldsymbol{I}^0(i, j)$ for convenience, has very low rank relative to its dimensions.

For the remaining of the this chapter, for convenience, we will treat the continuous 2D function and its sampled matrix form as the same, with the understanding that whenever we talk about distortion or transformation of a texture or an image, we mean a transformation in the 2D domain of its underlying continuous function. When only an image (a matrix of sampled values) is given, values of the function off the sampling grid can be obtained through any reasonable interpolation schemes.[6]

[4] The scale of the window needs to be large enough to meet this assumption.
[5] In other words, the resolution of the image cannot be too low.
[6] From our experience, the bicubic interpolation is good enough for most purposes.

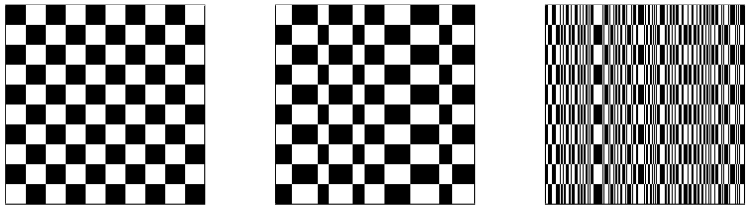


Figure 14.3. Different textural patterns: all three textures have exactly the same rank, but they go from purely regular, to nearly regular, and to almost irregular texture.

14.3 Structured Texture Inpainting

In this section, we will see how to automatically repair a structured texture when it is severely corrupted or occluded. From Chapters 4, we know if a texture $\boldsymbol{I}$ is a low-rank matrix, we can recover it even if only a small fraction of its entries (pixels) that are observable. Let Ω be the set of pixels given. The problem to recover the full texture image $\boldsymbol{I}^0$ is simply a low-rank matrix completion problem:

$$\min_{\boldsymbol{I}^0} \operatorname{rank}\left(\boldsymbol{I}^0\right) \quad \text{subj. to} \quad \boldsymbol{I}^0(i,j) = \boldsymbol{I}(i,j), \text{ for all } (i,j) \in \Omega. \tag{14.3.1}$$

Although being low-rank is a necessary condition for most regular, structured textures, it is certainly *not sufficient*. Figure 14.3 shows three images that have exactly the same rank. Obviously the first two are more smooth and regular than the third one. As discussed in the preceding section, a rank-r texture is a 2D function $\boldsymbol{I}^0(x,y)$, defined on $\mathbb{R}^2$. Then $\boldsymbol{I}^0(x,y)$ can be factored as

$$\boldsymbol{I}^0(x,y) = \sum_{i=1}^{r} u_i(x) v_i(y).$$

If $\boldsymbol{I}^0$ represents a more realistic regular or near regular pattern, it is typically piecewise smooth. Hence, the functions u_i and v_i are not arbitrary and they have additional structures. As we have discussed in earlier chapters of the book, piecewise smooth functions are typically sparse in certain transformed domain (say by Fourier or a wavelet transform).

So, in the discrete setting, the low-rank matrix $\boldsymbol{I}^0$ can be factorized as

$$\boldsymbol{I}^0 = \boldsymbol{U}\boldsymbol{V}^T,$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ can be represented as

$$\boldsymbol{U} = \boldsymbol{B}_1\boldsymbol{X}_1 \quad \text{and} \quad \boldsymbol{V} = \boldsymbol{B}_2\boldsymbol{X}_2$$

for some bases $(\boldsymbol{B}_1, \boldsymbol{B}_2)$. If the bases are properly chosen, both $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$ will be sufficiently sparse. Or equivalently, the matrix $\boldsymbol{W} = \boldsymbol{X}_1\boldsymbol{X}_2^T$ will be a sparse matrix, which has the same (low) rank as $\boldsymbol{I}^0$ since $\boldsymbol{I}^0 = \boldsymbol{B}_1\boldsymbol{W}\boldsymbol{B}_2^T$.

Hence, if we want the recovered image to be both low-rank and sparse (in certain transformed domain), we could modify the low-rank matrix completion problem (14.3.1) as follow to impose additional spatial structures:

$$\min_{\boldsymbol{I}^0,\boldsymbol{W}} \operatorname{rank}\left(\boldsymbol{I}^0\right) + \lambda\|\boldsymbol{W}\|_0 \;\; \text{s.t.} \;\; \mathcal{P}_\Omega(\boldsymbol{I}^0) = \mathcal{P}_\Omega(\boldsymbol{I}),\; \boldsymbol{I}^0 = \boldsymbol{B}_1\boldsymbol{W}\boldsymbol{B}_2^T, \tag{14.3.2}$$

where $\|\boldsymbol{W}\|_0$ denotes the number of non-zero entries in $\boldsymbol{W}$. That is, we aim to find the texture $\boldsymbol{I}^0$ of the lowest possible rank and the matrix $\boldsymbol{W} = \boldsymbol{B}_1^T\boldsymbol{W}\boldsymbol{B}_2$ with the fewest possible non-zero entries that agrees with the partial observation $\mathcal{P}_\Omega(\boldsymbol{I})$. Here, λ is a weighting parameter which trades off the rank and sparsity of the recovered image.

As we have learned from earlier chapters, in the above problem (14.3.2), both the rank function and the ℓ_0-norm are difficult to optimize directly. Instead, they can be replaced by their convex surrogates: the matrix nuclear norm $\|\boldsymbol{I}^0\|_*$ for $\operatorname{rank}\left(\boldsymbol{I}^0\right)$ and the ℓ_1-norm $\|\boldsymbol{W}\|_1$ for $\|\boldsymbol{W}\|_0$, respectively. Thus, we end up with the following optimization problem:

$$\min_{\boldsymbol{I}^0,\boldsymbol{W}} \|\boldsymbol{I}^0\|_* + \lambda\|\boldsymbol{W}\|_1 \text{ s.t. } \mathcal{P}_\Omega(\boldsymbol{I}^0) = \mathcal{P}_\Omega(\boldsymbol{I}),\; \boldsymbol{I}^0 = \boldsymbol{B}_1\boldsymbol{W}\boldsymbol{B}_2^T, \tag{14.3.3}$$

If we further assume that the bases $\boldsymbol{B}_1$ and $\boldsymbol{B}_2$ used are orthonormal, we have $\|\boldsymbol{I}^0\|_* = \|\boldsymbol{B}_1\boldsymbol{W}\boldsymbol{B}_2^T\|_* = \|\boldsymbol{W}\|_*$. The convex program (14.3.3) is equivalent to:

$$\min_{\boldsymbol{W}} \quad \|\boldsymbol{W}\|_* + \lambda\|\boldsymbol{W}\|_1 \;\; \text{s.t.} \;\; \mathcal{P}_\Omega(\boldsymbol{B}_1\boldsymbol{W}\boldsymbol{B}_2^T) = \mathcal{P}_\Omega(\boldsymbol{I}). \tag{14.3.4}$$

This formulation allow us to enforce that the recovered image textures be simultaneously low-rank and sparse in certain transformed domain.

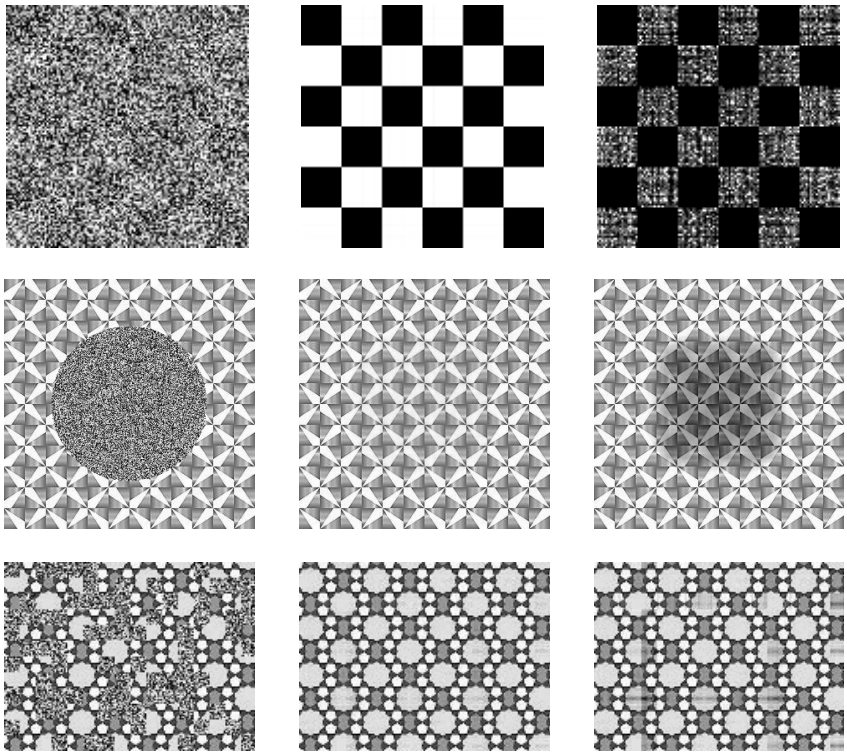
Notice that the above convex program (14.3.4) is different from the convex program we have encountered before in problems like PCP where the nuclear norm and ℓ^1 norm were for two different matrices. To utilize the same optimization techniques, we only have to introduce an auxiliary variable $\boldsymbol{L}$ to replace $\boldsymbol{W}$ in the low-rank term and render the variables separable:

$$\min_{\boldsymbol{L},\boldsymbol{W}} \|L\|_* + \lambda\|\boldsymbol{W}\|_1 \quad \text{s.t.} \;\; \boldsymbol{L} = \boldsymbol{W},\; \mathcal{P}_\Omega(\boldsymbol{B}_1\boldsymbol{W}\boldsymbol{B}_2^T) = \mathcal{P}_\Omega(\boldsymbol{I}). \tag{14.3.5}$$

The reader may recognize this program falls into the same class of programs as PCP that we have dealt with in Chapter 5, and they can be solved efficiently by methods introduced in Chapter 8.[7]

Example 14.3.1. *(Texture Inpainting). To demonstrate the importance of enforcing the sparse and low-rank prior together. We here conduct some texture impainting experiments on real images and compare the solution to*

[7] More detailed implementation of this particular program can be found in [Liang et al., 2012].



(a) Input image (b) Sparse + Low-rank (c) Low-rank only

Figure 14.4. **Qualitative comparison of sparse low-rank texture recovery and low-rank completion only.** The first row is a checkerboard texture with 91% randomly chosen pixels corrupted; the second row is a real texture with about 30% pixels occluded by a disk; and the third row is a real texture with about 40% pixels corrupted by random blocks.

the above program with that to low-rank matrix completion algorithm from Chapter 4.

Here we choose $\lambda = 0.001$*, and we use DCT for both* $\boldsymbol{B}_1$ *and* $\boldsymbol{B}_2$*. We test the recovery under three different types of corruptions: uniform random corruptions, one disk corruption, and random block corruptions on three representative low-rank textures: a checkerboard image (typically used in camera calibration) and two real texture images. The checkerboard is of precise rank* 2 *and the other two are full rank but approximately low-rank. From the completion results it is clear to see that the recovered results are significantly better by imposing both low-rank and sparse priors than low-rank alone.*

Almost all methods for image inpainting or completion need information about the support Ω of the corrupted regions (e.g., [Bertalmio et al., 2000], [Mairal et al., 2008b], [Fadili et al., 2009], etc.). 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. Similar to the Robust PCA problem in Chapter 5, we could model such corruptions with unknown support as a sparse error term $\boldsymbol{E}$:

$$\boldsymbol{I} = \boldsymbol{B}_1 \boldsymbol{W} \boldsymbol{B}_2^T + \boldsymbol{E}.$$

To recover the image $\boldsymbol{I}^0 = \boldsymbol{B}_1 \boldsymbol{W} \boldsymbol{B}_2^T$, we now only have to solve the following PCP like program:

$$\min_{\boldsymbol{W}} \quad \|\boldsymbol{W}\|_* + \lambda \|\boldsymbol{W}\|_1 + \alpha \|\boldsymbol{E}\|_1 \ \text{ s.t. } \mathcal{P}_\Omega(\boldsymbol{B}_1 \boldsymbol{W} \boldsymbol{B}_2^T + \boldsymbol{E}) = \mathcal{P}_\Omega(\boldsymbol{I}). \quad (14.3.6)$$

Notice that if we know nothing about the corruption areas, we only need to set Ω to be the entire image. Just like PCP, the above convex program will decompose the image into a low-rank component and a sparse one. Of course, the nonzero entries in estimated $\boldsymbol{E}$ can help us to further refine the support Ω. For instance, we could simply set:

$$\text{supp}(\boldsymbol{E}) \doteq \{(i,j) \in \Omega, \ |\boldsymbol{E}_{ij}| > \varepsilon, \} \quad (14.3.7)$$

for some threshold $\varepsilon > 0$. Or we could estimate the support of $\boldsymbol{E}$ using more sophisticated model to encourage additional structures such as spatial continuity [Zhou et al., 2009]. Once $\text{supp}(\boldsymbol{E})$ is known, we can exclude those corrupted entries from Ω (the support of presumably good entries).

We could further iterate between the image completion and support estimation:

$$\begin{aligned} (\boldsymbol{W}^i, \boldsymbol{E}^i) &= \text{argmin}_{\boldsymbol{W},\boldsymbol{E}} \|\boldsymbol{W}\|_* + \lambda \|\boldsymbol{W}\|_1 + \alpha \|\boldsymbol{E}\|_1 \\ &\qquad \text{subject to } \ \mathcal{P}_{\Omega^i}(\boldsymbol{B}_1 \boldsymbol{W} \boldsymbol{B}_2^T + \boldsymbol{E}) = \mathcal{P}_{\Omega^i}(\boldsymbol{I}), \\ \Omega^{i+1} &= \Omega^i \setminus \text{supp}(\boldsymbol{E}^{i+1}), \end{aligned} \quad (14.3.8)$$

where α is a weighting parameter between sparsity and low-rankness. We could continue the above process till convergence and obtain the repaired image $\boldsymbol{I}^* = \boldsymbol{B}_1 \boldsymbol{W}^* \boldsymbol{B}_2^T$. In practice, we notice a good side effect of adding the additional $\boldsymbol{E}$ term: It not only helps estimate support of the corrupted regions but also helps reduce noise on the repaired texture image $\boldsymbol{I}^*$.

Example 14.3.2. *(Texture Recovery) In this experiment, we conduct some comparison between the above method and some typical image completion methods used in highly engineered commercial systems: Patch Match (PM) used by Adobe Photoshop [Barnes et al., 2009, Barnes et al., 2010], Image*

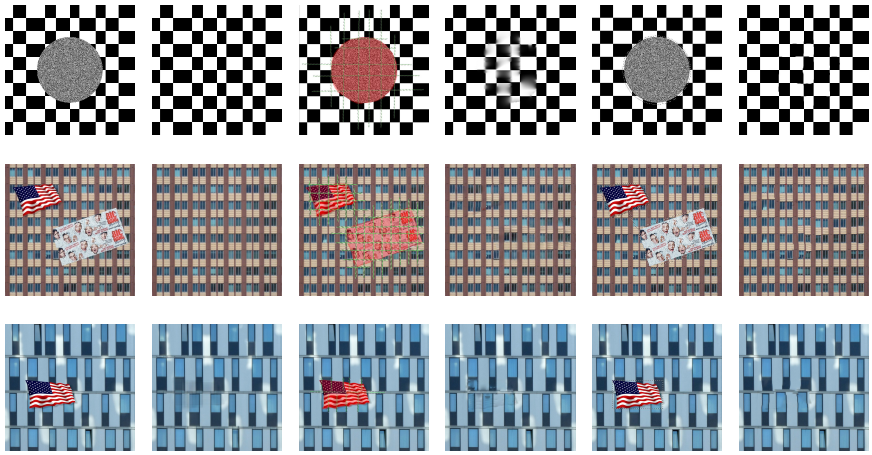


Figure 14.5. **Comparison results with Microsoft SP [Sun et al., 2005] and Adobe Photoshop [Barnes et al., 2009].** Columns 1-2: inputs and results of the structured texture recovery method; Columns 3-4: inputs and results of SP; Columns 5-6: inputs and results of Adobe Photoshop.

Completion with Structure Propagation (SP) developed by Microsoft [Sun et al., 2005]. Figure 14.5 shows the result on three different images: a simulated non-uniform low-rank texture, a uniform building facade, a somewhat less uniform building facade, and a real building facade image with perspective deformation, which correspond to four rows in Figure. 14.5, respectively.

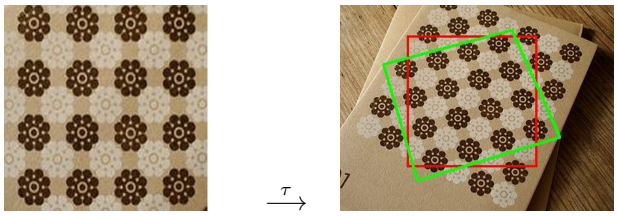
The other two methods all share the spirit of sample-based texture synthesis: they stitch sampled local patches together to ensure certain global statistical consistency. As these methods rely mostly on local statistics and structures, they tend to work on natural images or random textures too while our method does not. However, as we see from the results, when applied to completing or repairing regular or near regular *low-rank patterns, they often fail to preserve the global regularity accurately. The reason is partially because these methods normally do not or cannot exploit global structural information about the textures.*

Unlike the structured texture recovery method introduced here, these image completion systems typically require the user to mark out rather precisely the to-be-corrected region or regions (as the contours of the regions for Photoshop shown in the figure), and even to provide additional information about the structures to be recovered (such as suggested lines marked out in the red regions that required by the SP method). However, the structured texture recovery method does not need any knowledge about the support of the corrupted regions nor any information about the structure.

14.4 Transform Invariant Low-rank Textures

14.4.1 Deformed and Corrupted Low-rank Textures

Although a structured object that has regular or repetitive 2D or 3D textural patterns in space is often low-rank, its image $\boldsymbol{I}$ under an arbitrary camera viewpoint may exhibit much higher rank compared to its upright frontal view $\boldsymbol{I}^0(x, y)$. An example is illustrated in Figure 14.6. In order to extract the intrinsic low-rank textures from such deformed images, we need to carefully model the effect of deformation and see how to undo it correctly.



(a) Low-rank texture $\boldsymbol{I}^0$ (b) Its image $\boldsymbol{I}$ under a different viewpoint

Figure 14.6. An example of a transformed low-rank texture: The upright low-rank texture $\boldsymbol{I}^0$ on the left is associated with the region in the green window. The matrix associated with the region in the red window is clearly not low-rank.

Deformed Low-rank Textures.

Suppose a low-rank texture $\boldsymbol{I}^0(x, y)$ lies on certain surface in the scene. The image $\boldsymbol{I}$ is $\boldsymbol{I}^0$ taken from a certain camera viewpoint. We use τ to denote the transform where $\tau : \mathbb{R}^2 \to \mathbb{R}^2$ belongs to a certain Lie group $\mathbb{G}$ on $\mathbb{R}^2$ (We here only consider the case where τ is invertible). Hence $\boldsymbol{I} \circ \tau = \boldsymbol{I}^0$ or the image $\boldsymbol{I}$ can be viewed as transformed version of the original function $\boldsymbol{I}^0(x, y)$:

$$\boldsymbol{I}(x, y) = \boldsymbol{I}^0 \circ \tau^{-1}(x, y) = \boldsymbol{I}^0 \left(\tau^{-1}(x, y)\right),$$

If the texture is on a planar surface in the 3D space, under typical perspective camera model, one can assume $\mathbb{G}$ to be either the 2D affine group Aff(2), or the homography group $GL(3)$ acting linearly on the image domain. Nevertheless, in principle, the formulation also works for more general classes of domain deformations or camera projection models as long as they can be modeled well by a finite-dimensional parametric group [Zhang et al., 2011b, Zhang et al., 2011a].

Corrupted Low-rank Textures.

In addition to domain transformations, the observed image of the texture might be corrupted by pixel noise and occlusion. As before, we can model such nuisance by an error matrix $\boldsymbol{E}$ as follows:

$$\boldsymbol{I} = \boldsymbol{I}^0 + \boldsymbol{E}.$$

As a result, the image $\boldsymbol{I}$ might no longer be a low-rank texture. In the low-rank texture framework, we assume that only a small fraction of the image pixels are corrupted by gross errors. Hence, $\boldsymbol{E}$ will be a sparse matrix.

So the problem we are facing here is: *Given a possibly corrupted and deformed image of a low-rank texture:* $\boldsymbol{I} = (\boldsymbol{I}^0 + \boldsymbol{E}) \circ \tau^{-1}$, *can we recover both the intrinsic low-rank texture representation* $\boldsymbol{I}^0$ *and the domain transformation* $\tau \in \mathbb{G}$*?*

The answer to this problem is whether we are able to find solutions to the following optimization program:

$$\min_{\boldsymbol{I}^0, \boldsymbol{E}, \tau} \operatorname{rank}(\boldsymbol{I}^0) + \gamma \|\boldsymbol{E}\|_0 \quad \text{s.t.} \quad \boldsymbol{I} \circ \tau = \boldsymbol{I}^0 + \boldsymbol{E}. \tag{14.4.1}$$

That is, we aim to find the texture $\boldsymbol{I}^0$ of the lowest possible rank and the error $\boldsymbol{E}$ with the fewest possible nonzero entries that agrees with the observation $\boldsymbol{I}$ up to a domain transformation τ. Here, $\gamma > 0$ is a weighting parameter that trades off the rank of the texture versus the sparsity of the error. For convenience, we refer to the solution $\boldsymbol{I}^0$ to this problem as a *Transform Invariant Low-rank Texture* (TILT).[8]

14.4.2 The TILT Algorithm

As we studied in previous chapters, the rank function and the ℓ^0-norm in the original problem (14.4.1) are extremely difficult to optimize (in general NP-hard). However, under fairly broad conditions, they can be replaced by their convex surrogates: the matrix nuclear norm $\|\boldsymbol{I}^0\|_*$ for $\operatorname{rank}(\boldsymbol{I}^0)$ and the ℓ^1-norm $\|\boldsymbol{E}\|_1$ for $\|\boldsymbol{E}\|_0$, respectively. Thus, we end up with the following optimization problem:

$$\min_{\boldsymbol{I}^0, \boldsymbol{E}, \tau} \|\boldsymbol{I}^0\|_* + \lambda \|\boldsymbol{E}\|_1 \quad \text{s.t.} \quad \boldsymbol{I} \circ \tau = \boldsymbol{I}^0 + \boldsymbol{E}. \tag{14.4.2}$$

Dealing with domain deformation via linearization

Note that although the objective function in (14.4.2) is convex, the constraint $\boldsymbol{I} \circ \tau = \boldsymbol{I}^0 + \boldsymbol{E}$ is nonlinear in $\tau \in \mathbb{G}$, and hence the problem is not convex. We have seen a similar problem in Section 5.5. As we have discussed there, we may overcome this difficulty by linearizing the constraint

[8] By a slight abuse of terminology, we also refer to the procedure of solving the optimization problem as TILT.

Algorithm 14.1 (The TILT Algorithm)

INPUT: Input image $\boldsymbol{I} \in \mathbb{R}^{w \times h}$, initial transformation $\tau \in \mathbb{G}$ (affine or projective), and a weight $\lambda > 0$.

WHILE not converged **DO**

Step 1: Normalization and compute Jacobian:

$$\boldsymbol{I} \circ \tau \leftarrow \frac{\boldsymbol{I} \circ \tau}{\|\boldsymbol{I} \circ \tau\|_F}; \quad \nabla \boldsymbol{I} \leftarrow \frac{\partial}{\partial \zeta} \left(\frac{\mathrm{vec}(\boldsymbol{I} \circ \zeta)}{\|\mathrm{vec}(\boldsymbol{I} \circ \zeta)\|_F} \right) \Big|_{\zeta = \tau};$$

Step 2 (inner loop): Solve the linearized problem:

$$\begin{aligned} (\boldsymbol{I}^{0*}, \boldsymbol{E}^*, d\tau^*) \quad \leftarrow \quad & \arg\min_{\boldsymbol{I}^0, \boldsymbol{E}, d\tau} \ \|\boldsymbol{I}^0\|_* + \lambda \|\boldsymbol{E}\|_1 \\ & \text{s.t.} \ \ \boldsymbol{I} \circ \tau + \nabla \boldsymbol{I} \cdot d\tau = \boldsymbol{I}^0 + \boldsymbol{E}; \end{aligned}$$

Step 3: Update the transformation: $\tau \leftarrow \tau + d\tau^*$;

END WHILE

OUTPUT: Optimal solution $\boldsymbol{I}^{0*}$, $\boldsymbol{E}^*$, τ^* to problem (14.4.2).

around the current estimate and iterate, which is a typical technique to deal with nonlinearity in mathematical programming [Lucas and Kanade, 1981, Baker and Matthews, 2004]. If we approximate the nonlinear constraint up to its first order (with respect the deformation parameter τ), the constraint for the linearized version of the above program becomes

$$\boldsymbol{I} \circ \tau + \nabla \boldsymbol{I} \cdot d\tau = \boldsymbol{I}^0 + \boldsymbol{E}, \tag{14.4.3}$$

where $\nabla \boldsymbol{I}$ is the Jacobian (derivatives of the image with respect to the transformation parameters in τ).[9] The optimization problem in (14.4.2) reduces to

$$\min_{\boldsymbol{I}^0, \boldsymbol{E}, d\tau} \ \|\boldsymbol{I}^0\|_* + \lambda \|\boldsymbol{E}\|_1 \quad \text{s.t.} \quad \boldsymbol{I} \circ \tau + \nabla \boldsymbol{I} \cdot d\tau = \boldsymbol{I}^0 + \boldsymbol{E}. \tag{14.4.4}$$

The linearized problem above is a convex program and is amenable to efficient solution. Since the linearization is only a local approximation to the original nonlinear problem, we solve it iteratively in order to converge to a (local) minimum of the original non-convex problem (14.4.2). The resulting optimization scheme is summarized as Algorithm 14.1.

The iterative linearization scheme outlined above is a common technique in optimization to solve nonlinear problems. It can be shown that this kind of iterative linearization converges quadratically to a local minimum of the original non-linear problem. A complete proof is out of the scope of this

[9] Strictly speaking, $\nabla \boldsymbol{I}$ is a 3D tensor: it gives a vector of derivatives at each pixel whose length is the number of parameters in the transformation τ. When we "multiply" $\nabla \boldsymbol{I}$ with another matrix or vector, it contracts in the obvious way which should be clear from the context.

paper. We refer the interested reader to [Cromme, 1978, Jittorntrum and Osborne, 1980] and the references therein.

Solving the linearized inner loop program

To implement Algorithm 14.1 numerically, the most computationally expensive part is solving the inner loop convex program in Step 2. This can be cast as a semidefinite program and solved using conventional algorithms such as interior-point methods. However, as we discussed in Chapter 8, while interior-point methods have excellent convergence properties, they do not scale very well with the problem size. As TILT is a very very useful tool for computer vision, we here derive in more details a fast implementation based on the augmented Lagrangian method (ALM) vias *alternating direction method of multipliers* (ADMM), which was also covered in Chapter 8.

First, for the problem given in (14.4.4), its augmented Lagrangian is defined as:

$$\begin{aligned}\mathcal{L}_\mu(\boldsymbol{I}^0, \boldsymbol{E}, d\tau, \boldsymbol{Y}) \;\doteq\; & \|\boldsymbol{I}^0\|_* + \lambda\|\boldsymbol{E}\|_1 + \langle \boldsymbol{Y}, \boldsymbol{I}\circ\tau + \nabla\boldsymbol{I}\cdot d\tau - \boldsymbol{I}^0 - \boldsymbol{E}\rangle \\ & + \frac{\mu}{2}\left\|\boldsymbol{I}\circ\tau + \nabla\boldsymbol{I}\cdot d\tau - \boldsymbol{I}^0 - \boldsymbol{E}\right\|_F^2, \end{aligned} \tag{14.4.5}$$

where $\mu > 0$, $\boldsymbol{Y}$ is a Lagrange multiplier matrix, $\langle\cdot,\cdot\rangle$ denotes the matrix inner product. To optimize the above augmented Lagrangian, the augmented Lagrangian method require to solve the following steps iteratively:

$$\begin{aligned}(\boldsymbol{I}^0_{k+1}, \boldsymbol{E}_{k+1}, d\tau_{k+1}) &= \arg\min_{\boldsymbol{I}^0, \boldsymbol{E}, d\tau} \mathcal{L}_{\mu_k}(\boldsymbol{I}^0, \boldsymbol{E}, d\tau, \boldsymbol{Y}_k), \\ \boldsymbol{Y}_{k+1} &= \boldsymbol{Y}_k + \mu_k(\boldsymbol{I}\circ\tau + \nabla\boldsymbol{I}\cdot d\tau_{k+1} - \boldsymbol{I}^0_{k+1} - \boldsymbol{E}_{k+1}).\end{aligned}$$

Throughout the rest of the paper, we will always assume that $\mu_k = \rho^k\,\mu_0$ for some $\mu_0 > 0$ and $\rho > 1$, unless otherwise specified.

We only have to solve the first step of the above iterative scheme. In general, it is computationally expensive to minimize over all the variables $\boldsymbol{I}^0$, $\boldsymbol{E}$ and $d\tau$ simultaneously. So, we adopt a common strategy to solve it *approximately* by adopting an *alternating minimizing* strategy *i.e.,* minimizing with respect to $\boldsymbol{I}^0$, $\boldsymbol{E}$ and $d\tau$ one at a time:

$$\begin{aligned}\boldsymbol{I}^0_{k+1} &= \arg\min_{\boldsymbol{I}^0} \mathcal{L}_{\mu_k}(\boldsymbol{I}^0, \boldsymbol{E}_k, d\tau_k, \boldsymbol{Y}_k), \\ \boldsymbol{E}_{k+1} &= \arg\min_{\boldsymbol{E}} \mathcal{L}_{\mu_k}(\boldsymbol{I}^0_{k+1}, \boldsymbol{E}, d\tau_k, \boldsymbol{Y}_k), \\ d\tau_{k+1} &= \arg\min_{d\tau} \mathcal{L}_{\mu_k}(\boldsymbol{I}^0_{k+1}, \boldsymbol{E}_{k+1}, d\tau, \boldsymbol{Y}_k).\end{aligned} \tag{14.4.6}$$

Due to the special structure of our problem, each of the above optimization problems has a simple closed-form solution, and hence, can be solved in a single step. More precisely, the solutions to (14.4.6) can be expressed explicitly using the shrinkage operator as follows:

$$\begin{aligned}\boldsymbol{I}^0_{k+1} &\leftarrow \boldsymbol{U}_k\mathcal{S}_{\mu_k^{-1}}[\boldsymbol{\Sigma}_k]\boldsymbol{V}_k^T, \\ \boldsymbol{E}_{k+1} &\leftarrow \mathcal{S}_{\lambda\mu_k^{-1}}[\boldsymbol{I}\circ\tau + \nabla\boldsymbol{I}\cdot d\tau_k - \boldsymbol{I}^0_{k+1} + \mu_k^{-1}\boldsymbol{Y}_k], \\ d\tau_{k+1} &\leftarrow (\nabla\boldsymbol{I})^\dagger(-\boldsymbol{I}\circ\tau + \boldsymbol{I}^0_{k+1} + \boldsymbol{E}_{k+1} - \mu_k^{-1}\boldsymbol{Y}_k),\end{aligned} \tag{14.4.7}$$

Algorithm 14.2 (Inner Loop of TILT)

INPUT: The current (deformed and normalized) image $\boldsymbol{I} \circ \tau \in \mathbb{R}^{m \times n}$ and its Jacobian $\nabla \boldsymbol{I}$ against current deformation τ (from the outer loop), and $\lambda > 0$.

Initialization: $k = 0, \boldsymbol{Y}_0 = 0, \boldsymbol{E}_0 = 0, d\tau_0 = 0, \mu_0 > 0, \rho > 1$;

WHILE not converged **DO**

$(\boldsymbol{U}_k, \boldsymbol{\Sigma}_k, \boldsymbol{V}_k) = \text{SVD}\big(\boldsymbol{I} \circ \tau + \nabla \boldsymbol{I} \cdot d\tau_k - \boldsymbol{E}_k + \mu_k^{-1} \boldsymbol{Y}_k\big)$;

$\boldsymbol{I}^0_{k+1} = \boldsymbol{U}_k \mathcal{S}_{\mu_k^{-1}}[\boldsymbol{\Sigma}_k] \boldsymbol{V}_k^T$;

$\boldsymbol{E}_{k+1} = \mathcal{S}_{\lambda \mu_k^{-1}}[\boldsymbol{I} \circ \tau + \nabla \boldsymbol{I} \cdot d\tau_k - \boldsymbol{I}^0_{k+1} + \mu_k^{-1} \boldsymbol{Y}_k]$;

$d\tau_{k+1} = (\nabla \boldsymbol{I})^\dagger (-\boldsymbol{I} \circ \tau + \boldsymbol{I}^0_{k+1} + \boldsymbol{E}_{k+1} - \mu_k^{-1} \boldsymbol{Y}_k)$;

$\boldsymbol{Y}_{k+1} = \boldsymbol{Y}_k + \mu_k (\boldsymbol{I} \circ \tau + \nabla \boldsymbol{I} \cdot d\tau_{k+1} - \boldsymbol{I}^0_{k+1} - \boldsymbol{E}_{k+1})$;

$\mu_{k+1} = \rho \mu_k$;

END WHILE

OUTPUT: solution $(\boldsymbol{I}^0, \boldsymbol{E}, \Delta\tau)$ to problem (14.4.4).

where $\boldsymbol{U}_k \boldsymbol{\Sigma}_k \boldsymbol{V}_k^T$ is the SVD of $\big(\boldsymbol{I} \circ \tau + \nabla \boldsymbol{I} \cdot d\tau_k - \boldsymbol{E}_k + \mu_k^{-1} \boldsymbol{Y}_k\big)$, and $(\nabla \boldsymbol{I})^\dagger$ denotes the Moore-Penrose pseudo-inverse of $\nabla \boldsymbol{I}$.

We summarize the ADMM scheme for solving (14.4.4) as Algorithm 14.2. We note that the operations in each step of the algorithm are very simple with the SVD computation being the most computationally expensive step.[10]

As we see that it is essentially an nonlinear optimization problem to recover both the low rank texture and its deformation. The TILT algorithm relies on linearizing the nonlinear constraint locally and then iteratively solves the locally linearized version. Hence, in general, there is no guarantee if the algorithm converges to the globally optimal (usually the correct) solution. As studies in the literature [Zhang et al., 2010, Zhang et al., 2012] has shown, if the above algorithm is properly implemented, the range of convergence for typical deformations encountered in practice can be surprisingly large. For instance, for a typical checkerboard pattern tilted in front of a camera, the algorithm manages to converge correctly even if the tilting angle is around 50°! For details of implementing the TILT algorithm and a careful quantitative examination of the range of convergence for the TILT algorithm, the reader can refer to [Zhang et al., 2010, Zhang et al., 2012].

[10] Empirically, we notice that for larger window sizes (over 100×100 pixels), it is much faster to run the partial SVD instead of the full SVD, if the rank of the texture is known to be very low.

14.5 Applications of TILT

The above algorithm is derived for τ being an arbitrary (parametric) transform. In this section, we show how to apply the above algorithm to several typical types of transformations we often encounter in computer vision applications:

1. The low-rank texture is (approximately) on a planar surface and the camera is an ideal perspective projection. In this case, the deformation τ belongs to the group of general linear transforms on a plane (also known as the homography in the computer vision literature). The TILT algorithm allows us to recover the precise location and orientation of the plane in 3D relative to the camera.
2. The low-rank texture is on a generalized cylindrical surface. The TILT algorithm would allow use to recover both the 3D shape of the surface as well as its location and orientation relative to the camera.
3. The camera is not projective and its lens has certain nonlinear distortion. The images of a standard calibration rig (a planar checkerboard pattern) would allow us to recover the camera lens distortion.

14.5.1 Planar Low-rank Textures

If a low-rank texture $\boldsymbol{I}^0$ is on a planar surface, then at an arbitrary viewpoint, its image $\boldsymbol{I}$ (under ideal perspective projection) is related to the original (rectified) texture $\boldsymbol{I}^0$ by a homography τ [Ma et al., 2004], or formally known as a projective transformation. Figure 14.6 shows one such example. More precisely, let (u, v) be the coordinates of the image $\boldsymbol{I}$, and (x, y) be the coordinates of the original texture $\boldsymbol{I}^0$. If we represent both image planes with the homogeneous coordinates $(u, v, 1) \in \mathbb{R}^3$ and $(x, y, 1) \in \mathbb{R}^3$, respectively. Then the coordinates of a point on $\boldsymbol{I}^0$ and those of its (projective) image on $\boldsymbol{I}$ will be related by

$$\tau(x, y) = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \sim \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{14.5.1}$$

where "$\sim$" means equal up to a scale and $\boldsymbol{H} = [h_{ij}] \in \mathbb{R}^{3\times 3}$ is an invertible matrix belonging to the general linear group $\mathbb{GL}(3)$:

$$\mathbb{GL}(3) \doteq \left\{ \boldsymbol{H} \in \mathbb{R}^{3\times 3} \,\middle|\, \det(\boldsymbol{H}) \neq 0 \right\}. \tag{14.5.2}$$

However, there is a little caveat in this formulation. If we allow the transformation τ in the TILT algorithm to be free in the entire group $\mathbb{GL}(3)$, it could lead to a trivial solution in which the algorithm will choose a black region and a τ to blow it up to the size of $\boldsymbol{I}^0$ so that the value of the objective function will be nearly zero. To avoid such degenerate solutions,

one way is to fix the scale of the region which we like to rectify. We may restrict the transform to be in a subgroup of $\mathbb{GL}(3)$ with scale normalized:

$$\mathbb{GL}(3) \doteq \left\{ \boldsymbol{H} \in \mathbb{R}^{3\times 3} \,\middle|\, \det(\boldsymbol{H}) = 1 \right\}. \tag{14.5.3}$$

This imposes additional (nonlinear) constraints among the parameters of the transformation.[11] In practical implementation of the TILT algorithm for the projective case, to fix the scale, we may simply specify and fix two diagonal corners of the region we would like to rectify. Interested readers may find more implementation details in [Zhang et al., 2012]. Figure 14.7 shows some of the representative results of the TILT algorithm applied to low-rank textures on a plane. Notice that the rectification is typically accurate to the pixel level.

14.5.2 Generalized Cylindrical Surfaces

In man-made environments, structured objects do not always have planar surfaces. In many cases, the surface can be curved and cannot be approximated by a planar one, as the example in Figure 14.8 left shows. Clearly, if we approximate the surface by a plane outlined as the red window (adapted to the orientation of the facade), the texture will not be regular. Instead, the texture enclosed in the green window would be close to an ideal low-rank texture. However, to recover such low-rank texture, it would require us to recover the shape of the surface as well (in addition to the unknown camera projection in the planar case).

In this section, we see how TILT can be extended to rectify and recover low-rank texture on such curved surfaces in 3D space. Let us assume the image $\boldsymbol{I}(u, v)$ is a transformed version of low-rank texture $\boldsymbol{I}_0(x, y)$ wrapped on a curved surface C, as illustrated in Figure 14.8 right.

Presumably there exists a composite map from the intrinsic texture coordinate (x, y) to the image coordinate (u, v) as

$$g(x, y): \quad (x, y) \mapsto (u, v), \tag{14.5.4}$$

such that $\boldsymbol{I} \circ g = \boldsymbol{I}_0$ in the noise-free case. In this section, we will explain how to parametrize such a transformation g based on a generalized cylindrical surface model for the surface [Zhang et al., 2011a]. The model represents a very important family of 3D shapes as they describe majority of curved building facades or deformed texts on curved surfaces. Mathematically, a

[11] A systematical way to handle any additional constraints on the parameters of the transformation τ is to linearize it and then add the additional linear constraints to the Inner Loop of the TILT algorithm.

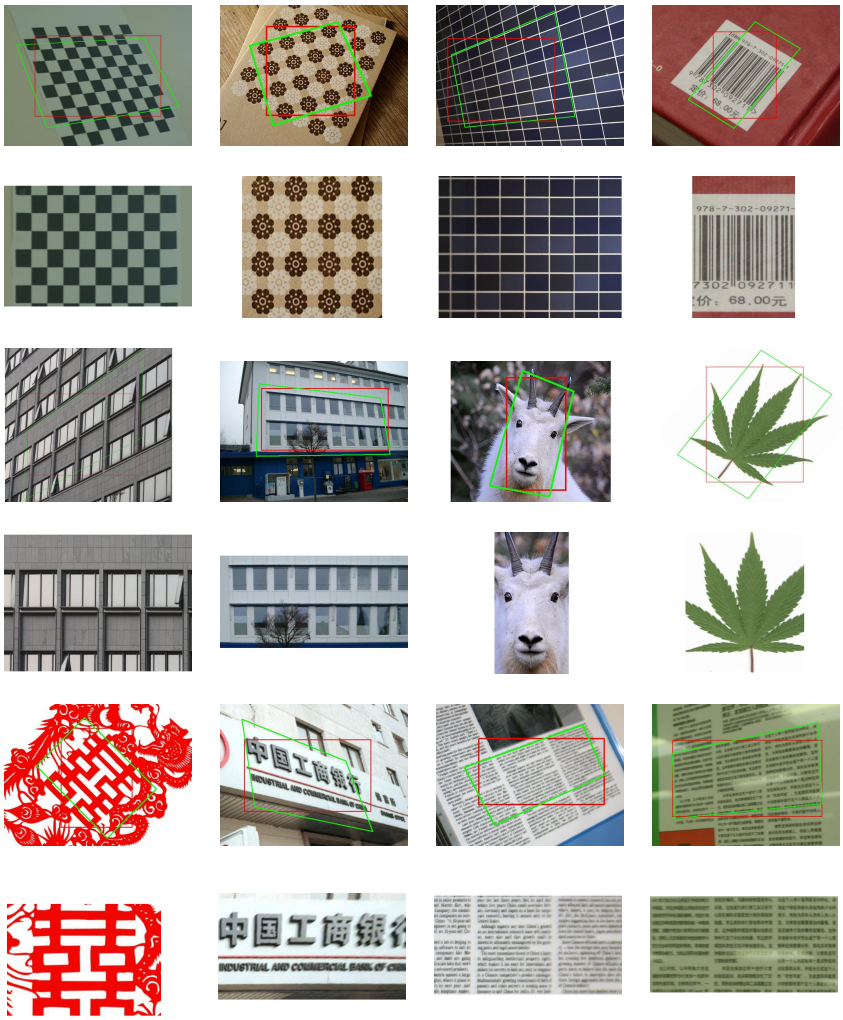


Figure 14.7. **Representative Results of TILT** on several categories of structured objects: textures with repetitive patterns, building facades, bar codes, characters and texts, bilateral symmetrical objects etc. In each case, the red window denotes the initial input of the TILT algorithm and the green window denotes the final converged output. The (matrix associated with the) green window is displayed to highlight the recovered low-rank texture.

generalized cylindrical surface can be described as

$$\boldsymbol{c}(s,t) = t\boldsymbol{p} + \boldsymbol{h}(s) \in \mathbb{R}^3, \tag{14.5.5}$$

where $s, t \in \mathbb{R}$, $\boldsymbol{p}, \boldsymbol{h}(s) \in \mathbb{R}^3$, and $\boldsymbol{p} \perp \partial\boldsymbol{h}(s)$.

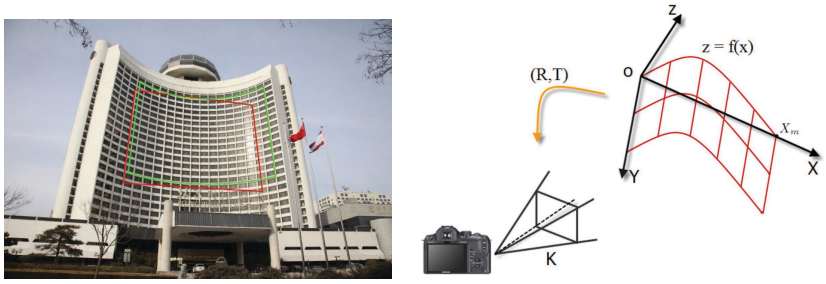


Figure 14.8. Left: An example of a curve building facade. Red window is a planar approximation to the surface; and green window outlines the true low-rank texture of the facade. Right: generalized cylindrical surface C viewed by a perspective camera K.

Without loss of generality, we may choose a 3D coordinate frame (X, Y, Z) for the surface such that the center O is on the surface and the Y-axis aligns with the direction of $\boldsymbol{p}$. If we limit our calculation within a "rectangular" section of the surface whose X-coordinate is in the interval $[0, X_m]$, then the expression of the function $\boldsymbol{h}(\cdot)$ can be simplified and uniquely determined by a scalar function $Z = f(X)$, as shown in Figure 14.8 right.

Without loss of generality, we may choose to parametrize the function $Z = f(X)$ by a polynomial up to degree $d + 2$, where typically $d \leq 4$ for most natural images. So an explicit expression of the surface can be written as:

$$Z = f_c(X) = X(X - X_m) \sum_{i=0}^{d} a_i X_i, \tag{14.5.6}$$

where we use $\boldsymbol{c} \doteq \{a_0, a_1, \ldots, a_d\}$ to denote the collection of surface parameters. Further note that when all a_i's are zero, the surface reduces to a planar surface $Z = 0$ in 3D as considered in the previous section.

For any point (x, y) in the rectified and flattened texture coordinates, we need to find its 3D coordinates (X_c, Y_c, Z_c) on the cylindrical surface C. We can calculate the geodesic distance from the origin O to $(X_m, 0, 0)$ on the surface as

$$L_c \doteq \int_0^{X_m} \sqrt{1 + f_c'(X)^2} dX. \tag{14.5.7}$$

The the following set of equations uniquely determine the wrapping map between (x, y) and (X_c, Y_c, Z_c):

$$\begin{cases} x &= \frac{X_m}{L_c} \int_0^{X_c} \sqrt{1 + f_c'(X)^2} dX, \\ y &= Y_c, \\ Z_c &= f_c(X_c). \end{cases} \tag{14.5.8}$$

Finally, suppose we are given a perspective camera model with intrinsic parameters $\boldsymbol{K} = \begin{bmatrix} f_x & \alpha & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{3\times 3}$,[12] and its relative position with respect to the surface coordinate frame is described by an unknown Euclidean transform $(\boldsymbol{R}, \boldsymbol{T})$, where $\boldsymbol{R} \in \mathbb{R}^{3\times 3}$ is a rotation matrix and $\boldsymbol{T} \in \mathbb{R}^3$ is a translation vector. Then a point with 3D coordinates (X_c, Y_c, Z_c) is mapped to the image pixel coordinates (u, v) according to the following relationship:

$$\begin{bmatrix} x_n \\ y_n \end{bmatrix} = \begin{bmatrix} \frac{R_{11}X_c + R_{12}Y_c + R_{13}Z_c + T_1}{R_{31}X_c + R_{32}Y_c + R_{33}Z_c + T_3} \\ \frac{R_{21}X_c + R_{22}Y_c + R_{23}Z_c + T_2}{R_{31}X_c + R_{32}Y_c + R_{33}Z_c + T_3} \end{bmatrix}, \tag{14.5.9}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \boldsymbol{K} \begin{bmatrix} x_n \\ y_n \\ 1 \end{bmatrix} = \begin{bmatrix} f_x x_n + \alpha y_n + o_x \\ f_y y_n + o_y \\ 1 \end{bmatrix}. \tag{14.5.10}$$

Therefore the transformation g from the texture coordinates (x, y) to the image coordinates (u, v) is a composition of the following mappings specified above:

$$g : (x, y) \to (X_c, Y_c, Z_c) \to (x_n, y_n) \to (u, v). \tag{14.5.11}$$

The parameters needed to specify g include the parameters for the surface c, the camera pose $(\boldsymbol{R}, \boldsymbol{T})$ (and the camera intrinsic parameters $\boldsymbol{K}$ if unknown).

In order to recover the low-rank component $\boldsymbol{I}^0$ and the sparse error component $\boldsymbol{E}$, we can now solve the following optimization problem:

$$\min_{\boldsymbol{c},\boldsymbol{R},\boldsymbol{T},\boldsymbol{I}^0,\boldsymbol{E}} \|\boldsymbol{I}^0\|_* + \lambda\|\boldsymbol{E}\|_1 \quad \text{subj. to} \quad \boldsymbol{I} \circ g = \boldsymbol{I}^0 + \boldsymbol{E}. \tag{14.5.12}$$

As we have done in the TILT algorithm, this nonlinear problem can be estimated iteratively by solving the linearized version as:

$$\min_{dg,\boldsymbol{I}^0,\boldsymbol{E}} \|\boldsymbol{I}^0\|_* + \lambda\|\boldsymbol{E}\|_1 \quad \text{subj. to} \quad \boldsymbol{I} \circ g + \boldsymbol{J}_g dg = \boldsymbol{I}^0 + \boldsymbol{E} \tag{14.5.13}$$

where $\boldsymbol{J}_g$ is the Jacobian matrix of the image with respect to both the unknown general cylindrical surface parameters $\boldsymbol{c}$ and the unknown Euclidean transform $(\boldsymbol{R}, \boldsymbol{T})$ (and the camera calibration $\boldsymbol{K}$ if unknown too), and dg is the differential of these unknown variables. For a more detailed implementation, please refer to [Zhang et al., 2011a]. Figure 14.9 shows some real examples and results of curved low-rank textures unwrapped and recovered by the TILT algorithm based on the transformation g described above.

[12] Be aware that here f_x, f_y stand for focus length and are not to be confused with the curve function f_c above. In practice, the intrinsic parameters can be well approximated from the EXIF information in the image file recorded by modern digital cameras. For more details, please refer to [Zhang et al., 2011a].

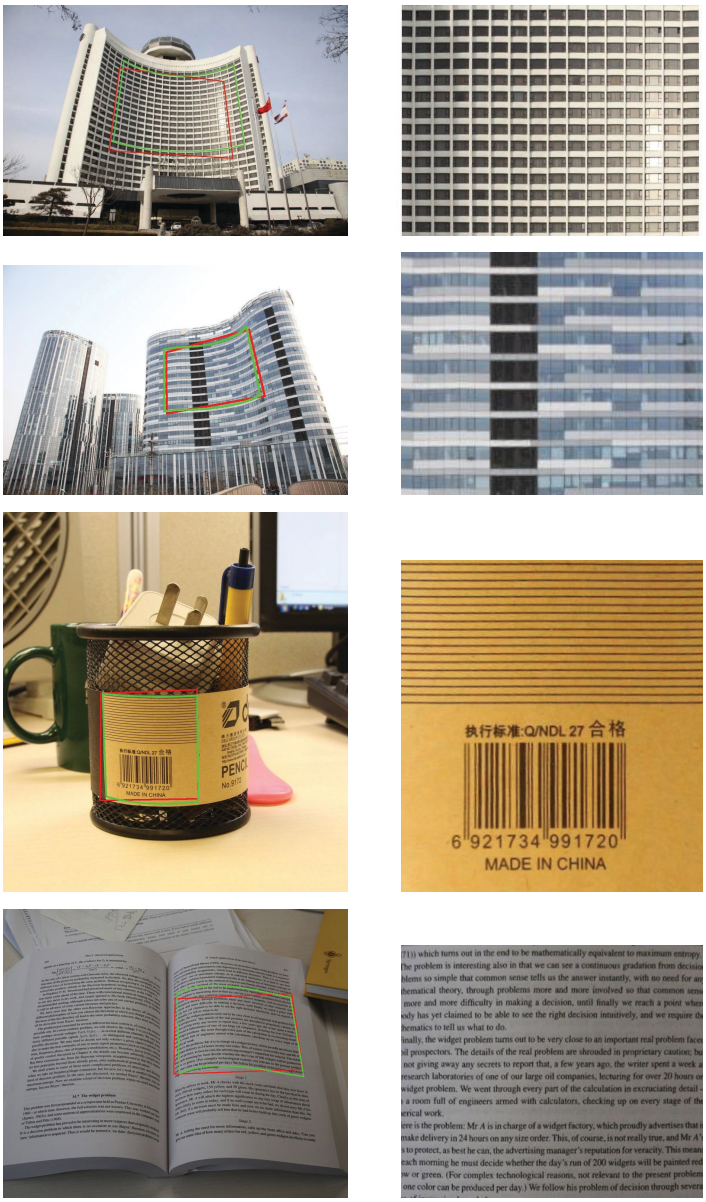


Figure 14.9. Unwrapping low-rank texture on curved surfaces based on a generalized cylindrical surface model. Left: Input image. The red bounding box indicates the manually labeled initial position of the texture region. The green bounding box indicates the recovered texture surface. Right: Unwrapped low-rank texture.

Figure 14.10. Left: typical image of a fisheye camera. Right: image of a perspective camera.

14.5.3 Calibrating Camera Lens Distortion

In the above planar and curved surface cases, we have always assumed that the image of a low-rank texture is taken by a (calibrated) camera which can be represented by an ideal perspective projection. However, with the proliferation of low-cost cameras, many cameras (and their images) we encounter in practice are not carefully calibrated by the manufacturer; and some cameras are deliberated made with different projection models from the perspective one (in order to maximize the use of the imaging sensors and increase the field of view), such as the fisheye cameras or omnidirectional cameras. Figure 14.10 left shows an example of an image taken by a fisheye camera. Figure 14.10 right shows an image of the same building by an ideal perspective camera. Notice that for the later case, there is no distortion caused by the lens and all straight lines in the scene are straight in the image.

Camera calibration is arguably the most crucial task for any applications that requires the computer or machine to perceive and interact with the 3D world through a camera (such as 3D reconstruction, mapping, navigation, and manipulation etc.) When calibrating a camera (with respect to certain world coordinate frame), in addition to the aforementioned *lens distortion*, we also need to calibrate the *intrinsic parameters* $\boldsymbol{K}$ for its perspective projection and the *extrinsic parameters* $(\boldsymbol{R}, \boldsymbol{T})$ for the camera viewpoint. For conventional calibration methods such as the popular toolbox [Bouguet,], one needs to take a few images of a calibration rig (usually a planar checkerboard pattern with known geometry [Zhang, 2000]). The corners of the checkerboard then need to be carefully marked out for calibration. Figure 14.11 shows an example.

As we see, the calibration rig is typically a regular pattern hence is low-rank. The imaging process of an uncalibrated camera is a sequence of mappings that transform the low-rank texture to the image plane. Instead

Figure 14.11. Left two: Images of a typical calibration rig. Right: Corners need to be marked (or detected) for conventional calibration methods.

of marking the corners manually which is tedious and time-consuming,[13] in principle we could utilize the low-rankness of the calibration pattern and use the TILT algorithm to automatically estimate all unknown parameters associated with the imaging process.

Similar to the previous section, we here give a brief description of the sequence of mappings involved in the imaging process of an uncalibrated camera, which can then be used in customizing the TILT algorithm for calibration purposes. For a more careful and complete description of camera models, the reader may refer to [Hartley and Zisserman, 2000, Ma et al., 2004].

We first briefly describe the common mathematical model used for camera calibration and introduce notation used in this paper. We use a vector $\boldsymbol{P} = [X_0, Y_0, Z_0]^T \in \mathbb{R}^3$ to denote the 3D coordinates of a point in the world coordinate frame, use $\boldsymbol{p}_n = [x_n, y_n]^T \in \mathbb{R}^2$ to denote its projection on the canonical image plane in the camera coordinate frame. For convenience, we denote the homogeneous coordinates of a point p as $\tilde{\boldsymbol{p}} = \left[\begin{smallmatrix}\boldsymbol{p}\\1\end{smallmatrix}\right]$.

As before, we use $\boldsymbol{R} \in \mathbb{SO}(3)$ and $\boldsymbol{T} \in \mathbb{R}^3$ to denote the rotation and translation from the world coordinate frame to the camera frame – so-called extrinsic parameters.[14] So we have

$$\tilde{\boldsymbol{p}}_n \sim \boldsymbol{R}\boldsymbol{P} + \boldsymbol{T}.$$

If the lens of the camera is distorted, on the image plane, the coordinates of a point $\boldsymbol{p}_n$ may be transformed to a different one, denoted as $\boldsymbol{p}_d = [x_d, y_d]^T \in \mathbb{R}^2$. A very commonly used general mathematical model for this distortion $D : \boldsymbol{p}_n \mapsto \boldsymbol{p}_d$ is given by a polynomial distortion model by

[13] Automatically detecting the corner features is a seemingly simple but remains a difficult problem that is not yet entirely solved under general conditions.

[14] As in [Ma et al., 2004], the rotation $\boldsymbol{R}$ can be parameterized by a vector $\boldsymbol{\omega} = [\omega_1, \omega_2, \omega_3]^T \in \mathbb{R}^3$ using the Rodrigues formula: $\boldsymbol{R}(\boldsymbol{\omega}) = \boldsymbol{I} + \sin\|\boldsymbol{\omega}\|\frac{\hat{\boldsymbol{\omega}}}{\|\boldsymbol{\omega}\|} + (1 - \cos\|\boldsymbol{\omega}\|)\frac{\hat{\boldsymbol{\omega}}^2}{\|\boldsymbol{\omega}\|^2}$, where $\hat{\boldsymbol{\omega}}$ denotes the 3×3 matrix form of the rotation vector $\boldsymbol{\omega}$, defined as $\hat{\boldsymbol{\omega}} = [0, -\omega_3, \omega_2; \omega_3, 0, -\omega_1; -\omega_2, \omega_1, 0] \in \mathbb{R}^{3\times 3}$.

neglecting any higher-order terms as below [Brown, 1971]:

$$\begin{aligned} r &\doteq \sqrt{x_n^2 + y_n^2}, \\ f(r) &\doteq 1 + k_c(1)r^2 + k_c(2)r^4 + k_c(5)r^6, \\ \boldsymbol{p}_d &= \begin{bmatrix} f(r)x_n + 2k_c(3)x_n y_n + k_c(4)(r^2 + 2x_n^2) \\ f(r)x_n + 2k_c(4)x_n y_n + k_c(3)(r^2 + 2y_n^2) \end{bmatrix}. \end{aligned} \tag{14.5.14}$$

Notice that this model has a total of five unknowns $k_c(1), \ldots, k_c(5) \in \mathbb{R}$. If there is no distortion, simply set all $k_c(i)$ to be zero, and then it becomes $\boldsymbol{p}_d = \boldsymbol{p}_n$.

The intrinsic matrix $\boldsymbol{K} \in \mathbb{R}^{3\times 3}$ represents a linear transformation of points on the image plane to their pixel coordinates, denoted as $\boldsymbol{p} = [u, v]^T \in \mathbb{R}^2$. $\boldsymbol{K}$ also has five unknowns; the focal length along x and y-axes f_x and f_y, skew parameter θ, and coordinates of the principle point (o_x, o_y). In the matrix form, it is described as

$$\boldsymbol{K} \doteq \begin{bmatrix} f_x & \theta & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \quad \in \mathbb{R}^{3\times 3}. \tag{14.5.15}$$

With all the notation, the overall imaging process of a point $\boldsymbol{P}$ in the world to the camera pixel coordinates $\boldsymbol{p}$ by a pinhole camera can be described as:

$$\tilde{\boldsymbol{p}} = \boldsymbol{K}\tilde{\boldsymbol{p}}_d = \boldsymbol{K} \circ D(\tilde{\boldsymbol{p}}_n); \quad \lambda\tilde{\boldsymbol{p}}_n = \boldsymbol{R}\boldsymbol{P} + \boldsymbol{T}, \tag{14.5.16}$$

where λ is the depth of the point. If there is no lens distortion ($\boldsymbol{p}_d = \boldsymbol{p}_n$), the above model reduces the typical perspective projection with an uncalibrated camera: $\lambda\tilde{\boldsymbol{p}} = \boldsymbol{K}(\boldsymbol{R}\boldsymbol{P} + \boldsymbol{T})$.

For compact presentation, later in this paper, we will let τ_0 denote the intrinsic parameters and lens distortion parameters all together. When we take multiple, say N, images to calibrate the intrinsic parameters and the lens distortion, we use τ_i $(i = 1, 2, \ldots, N)$ to denote the extrinsic parameters $\boldsymbol{R}_i$ and $\boldsymbol{T}_i$ for the i-th image. By a slight abuse of notation, we will occasionally use τ_0 to represent the combined transformation of $\boldsymbol{K}$ and D acting on the image domain, i.e., $\tau_0(\cdot) = \boldsymbol{K} \circ D(\cdot)$, and use τ_i $(i = 1, 2, \ldots, N)$ to represent the transforms from the world frame to each individual image plane. Using this notation, each image $\boldsymbol{I}_i$ and the calibration rig (low-rank texture) $\boldsymbol{I}_0$ are related by:

$$\boldsymbol{I}_i \circ (\tau_0 \circ \tau_i) = \boldsymbol{I}_0 + \boldsymbol{E}_i, \quad i = 1, 2, \ldots, N, \tag{14.5.17}$$

where we use a sparse error term $\boldsymbol{E}_i$ to model possible occlusion or corruption introduced in the imaging process.

It seems that we now can use TILT to estimate all the transformation parameters in τ_0 and τ_i without using any marked feature points. However, there is one caveat: as we have discussed in the beginning of the chapter in Section 14.2, there is some ambiguity in the notion of a low-rank texture, a

scaled or translated version of the same texture would have the same rank. Hence, if we use TILT to each individual image $\boldsymbol{I}_i$, the $\boldsymbol{I}_0$ so recovered might be scaled or translated version to one another. More precisely, if we solve the following robust rank-minimization problems individually:

$$\min \|\boldsymbol{L}_i\|_* + \lambda\|\boldsymbol{E}_i\|_1, \quad \text{s.t.} \quad \boldsymbol{I}_i \circ (\tau_0 \circ \tau_i) = \boldsymbol{L}_i + \boldsymbol{E}_i, \tag{14.5.18}$$

with $\boldsymbol{L}_i, \boldsymbol{E}_i, \tau_i$ and τ_0 as unknowns. Then we can only expect $\boldsymbol{L}_i$ to recovers the low-rank pattern $\boldsymbol{I}_0$ up to a translation and scaling in each axis, i.e.,

$$\boldsymbol{L}_i = \boldsymbol{I}_0 \circ \tau, \quad \text{where} \quad \tau = \begin{bmatrix} a & 0 & t_1 \\ 0 & b & t_2 \\ 0 & 0 & 1 \end{bmatrix}. \tag{14.5.19}$$

There are many possible ways we can use all the N images together and try to estimate a common solution for τ_0 and $\boldsymbol{I}_0$. The most straightforward way is to lump all the objective functions together and enforce that the recovered $\boldsymbol{L}_i$ are all the same: Therefore, the natural optimization problem associated with this problem becomes

$$\begin{aligned} \min \quad & \sum_{i=1}^{N} \|\boldsymbol{L}_i\|_* + \|\boldsymbol{E}_i\|_1, \\ \text{s.t.} \quad & \boldsymbol{I}_i \circ (\tau_0 \circ \tau_i) = \boldsymbol{L}_i + \boldsymbol{E}_i, \ \boldsymbol{L}_i = \boldsymbol{L}_j. \end{aligned} \tag{14.5.20}$$

One can use optimization techniques similar to that of TILT to solve the above optimization problem, such as ALM and ADMM. However, having too many constraining terms affects the convergence of such algorithms.

To relax the equality constraint $\boldsymbol{L}_i = \boldsymbol{L}_j$, we may only need to require they are correlated. So an alternative is to concatenate all the recovered low-rank textures as submatrices of a joint low-rank matrix:

$$\begin{aligned} & \boldsymbol{L}_c \doteq [\boldsymbol{L}_1, \boldsymbol{L}_2, \ldots, \boldsymbol{L}_N], \ \boldsymbol{L}_r \doteq [\boldsymbol{L}_1^T, \boldsymbol{L}_2^T, \ldots, \boldsymbol{L}_N^T], \\ & \boldsymbol{E} \doteq [\boldsymbol{E}_1, \boldsymbol{E}_2, \ldots, \boldsymbol{E}_N]. \end{aligned} \tag{14.5.21}$$

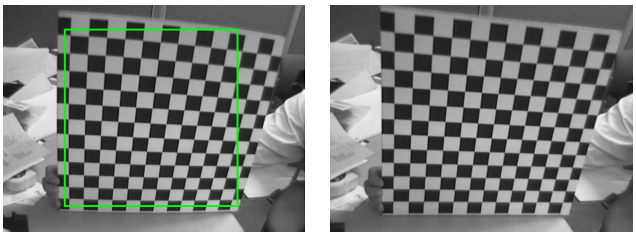
Then we try to simultaneously align the columns and rows of $\boldsymbol{L}_i$ by minimizing the ranks of $\boldsymbol{L}_c$ and $\boldsymbol{L}_r$:

$$\begin{aligned} \min \quad & \|\boldsymbol{L}_c\|_* + \|\boldsymbol{L}_r\|_* + \lambda\|\boldsymbol{E}\|_1, \\ \text{s.t.} \quad & \boldsymbol{I}_i \circ (\tau_0 \circ \tau_i) = \boldsymbol{L}_i + \boldsymbol{E}_i, \end{aligned} \tag{14.5.22}$$

with $\boldsymbol{L}_i, \boldsymbol{E}_i, \tau_0, \tau_i$ as unknowns. Notice that, by comparing to Eq. (14.5.20), the new optimization program has just half number of constraints and hence is easier to solve. In addition, it is insensitive to illumination and contrast change across different images as it does not require the recovered the images $\boldsymbol{L}_i$ to be equal in values.

It can be shown (see [Zhang et al., 2011b]) that under general conditions, when the number of images $N \geq 5$, then the optimal solution to the above program will be unique and

$$\tau_0^* = \tau_0, \quad \boldsymbol{K}^* = \boldsymbol{K}, \quad \boldsymbol{R}_i^* = \boldsymbol{R}_i, \quad i = 1, \ldots N.$$



(a) Input image with initial window (b) Lens distortion removed

Figure 14.12. Calibration from a single image in the Toolbox.

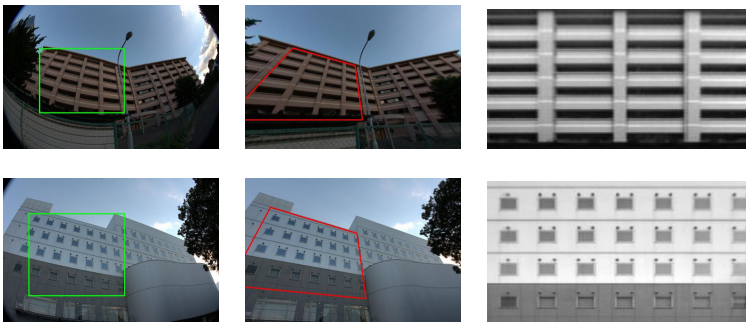


Figure 14.13. Rectify fisheye images with significant lens distortion. **Left:** input images with selected initialization windows (green); **Middle:** rectified images with final converged windows (red); **Right:** rectified low-rank textures.

In practice, the method actually works with as few as a single ($N = 1$) image (of low-rank texture). To calibrate the camera from a single image, we have to work with fairly strong assumptions, say that the principal point (o_x, o_y) is known (and simply set as the center of the image) and the pixel is square $f_x = f_y$. Then from the image, one can calibrate the focal length as well as eliminating the lens distortion k_c's. Figure 14.12 shows an example with an image given in the standard toolbox. As we see in Figure 14.12(b), the radial distortion is completely removed by the TILT algorithm.

Notice that the low-rank texture based calibration method does not require precise geometry about the calibration rig. Hence one does not have to use a checkerboard pattern and in principle can utilize any low-rank structures (such as a building facade) for calibration purposes. Figure 14.13 shows two examples of using the TILT algorithm to estimate and correct the radial lens distortion of a fisheye camera (from a single image).

14.6 Notes and References

The story presented in this chapter follows from the original work of [Zhang et al., 2010, Zhang et al., 2012] on the TILT method and its extensions to curved surface [Zhang et al., 2011a], camera calibration [Zhang et al., 2011b], and texture inpainting [Liang et al., 2012].

Transformed sparse or low-rank models have also been explored and utilized in the work for sparsity-based face alignment and recognition [Wagner et al., 2009, Wagner et al., 2012] and for low-rank based multiple-image alignment method RASL [Peng et al., 2012].

Exercises

14.1 (New Exercise)**.**