Image Stitching

Shree K. Nayar

Lecture: FPCV-2-4

Module: Features

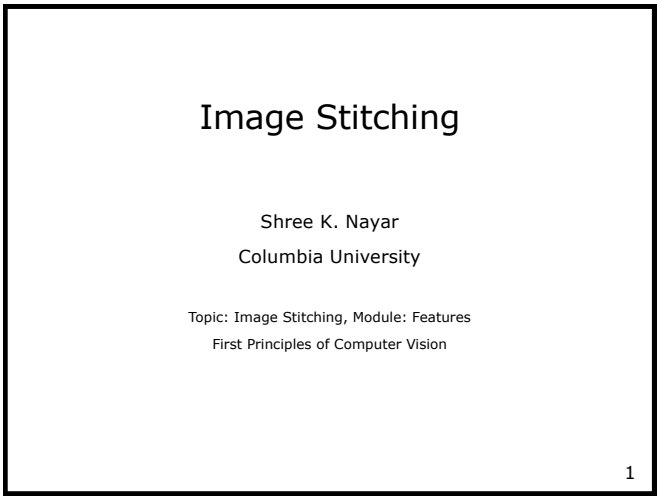
Series: First Principles of Computer Vision

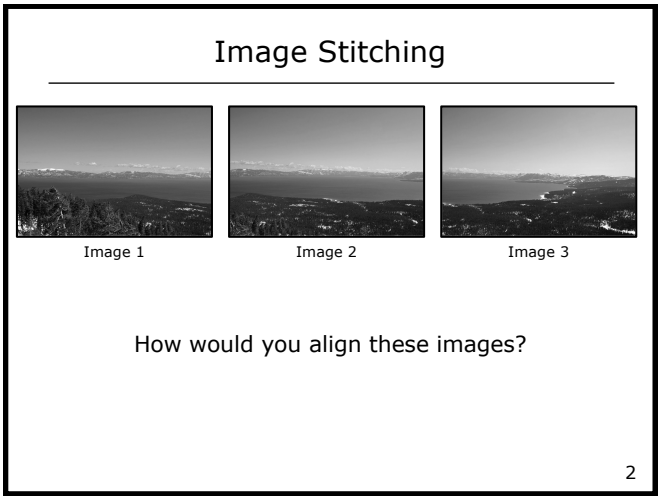
Computer Science, Columbia University

January, 2025

FPCV Channel

FPCV Website

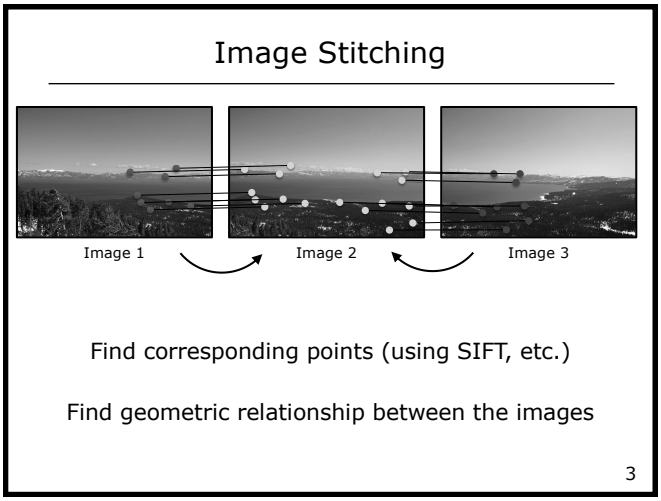




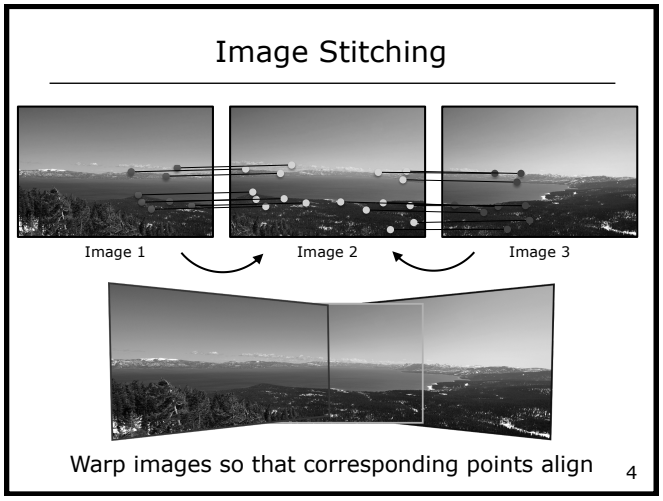
In this lecture, we describe image stitching. A series of images of a 3D scene are obtained by rotating a camera. As the images are taken, it must be ensured that the fields of view of consecutive images overlap. We can then automatically stitch these images to create a wide-angle panorama. This is a technology that is now available on most smartphones. Image stitching is also popular in other domains, including medical imaging and remote sensing.

Let us begin by describing the steps involve in image stitching. Given the three images shown on the right, we need to first align them with respect to each other.

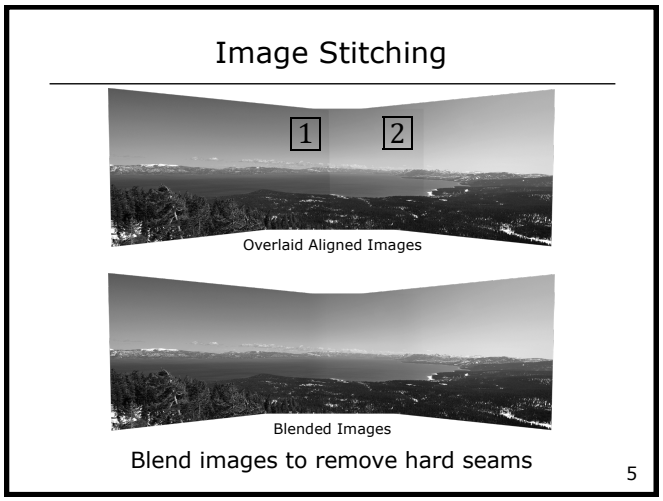
For alignment, a feature detector, such as the SIFT detector, is applied to the images to extract features, which are shown as dots on the images. Based on the resulting SIFT descriptors, we can then match features between the images to obtain the pairs of matching features shown by the lines overlaid on the images. The next step is to determine the geometric relationship between the images. In other words, we seek to find the transformation that takes one image and warps it to the coordinate frame of the other image. Later in the lecture, we are going to describe what that transformation is and how one computes it.

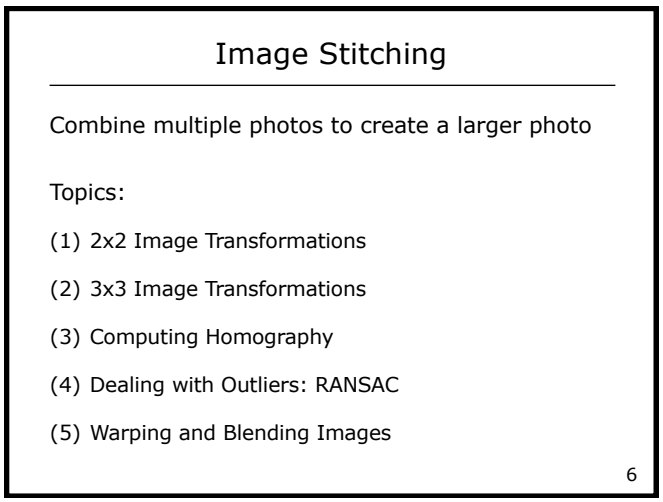


Once the transformations between the images has been computed, one of the images can be designated as the reference images, and all the other images can be mapped to the coordinate frame of the reference image. The result is a stack of overlapping images, as shown at the bottom. We are not quite done with image stitching yet, as we have one last problem to solve—removing the seams.



A scene point will likely produce slightly different brightnesses in the different images it appears in. This is due to various effects, including, exposure variations between the images, change of lighting between the capture of the images, and spatial camera response variations due to effects such as vignetting. Thus, we are almost always guaranteed to end up with visible seams between overlapping images, like 1 and 2. Our goal is to remove these seams to obtain a single, clean image. That brings us to the topic of blending images. We will develop a simple blending algorithm that helps remove seams to create a single smooth panorama like the one shown at the bottom.





We will begin this lecture by discussing image transformations. In particular, we will identify the image transformation needed to warp an image and align it with another image. We begin with simple 2x2 image transformation matrices, and describe the different transformations possible with just four parameters. We will argue that a 2x2 matrix cannot achieve the type of transformation needed to align perspective images of a 3D scene. To this end, we explore 3x3 image transformations. In particular, we describe the projective transformation which is given by a 3x3 matrix called the *homography*. We develop an algorithm for computing the homography between two images, given pairs of matching points in the images.

When we take two images and apply the SIFT detector to find matching features, we will end up with some invalid matches. We can have two features that match because they have very similar local appearances, but they do not actually correspond to the same point in the scene. In other words, our set of matching features is going to have inliers, which are valid pairs, as well as outliers, which are invalid pairs. We need to come up with an approach for dealing with the outliers. That brings us to a clever and useful algorithm called RANSAC. We will show that if the outliers don't dominate the inliers in a set of matching pairs of points, we can compute a valid homography that is unaffected by the outliers.

Finally, when we warp the images to a single coordinate frame, we will invariably end up with differences in brightness between the images. We will present a simple blending algorithm for removing these differences and creating a seamless panorama.

2x2 Image Transformations

Shree K. Nayar
Columbia University

Topic: Image Stitching, Module: Features
First Principles of Computer Vision

7

Image Manipulation

Image Filtering: Change range (brightness)

$$g(x,y) = T_r(f(x,y))$$

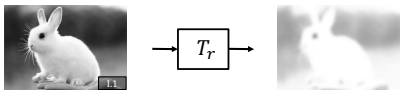
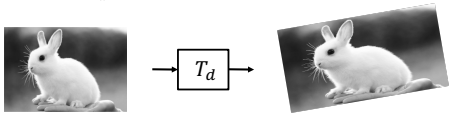


Image Warping: Change domain (location)

$$g(x,y) = f\big(T_d(x,y)\big)$$

Transformation T_d is a coordinate changing operator

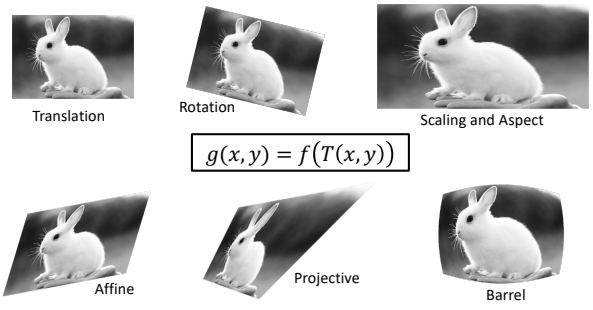


8

There are two general classes of image transformations. The first is image filtering, which we discussed during our lectures on image processing. In this case, the transformation acts on the range of brightness values in the image and does not change the shape of the image. Techniques such as pixel processing and convolution lie within this class of transformations. The second class of image transformations is what we will call warping. In this case, the transformation acts on the domain of the image, that is, the *x* and *y* coordinates of the image, to change the "shape" of the image. We use T_r to denote a range transformation and T_d to denote a domain transformation. In the context of image stitching, we are only interested in domain transformations.

Shown here are some geometric transformations that can be applied to an image. Each of these is a domain transformation, or a warp, and is made possible using a very small number of parameters.

Global Warping/Transformation



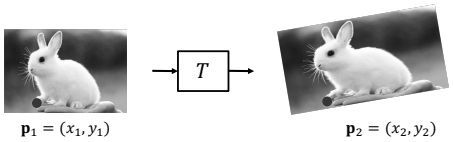
$$g(x,y) = f\big(T(x,y)\big)$$

Transformation T is the same over entire domain
Often can be described by just a few parameters

9

First, let us start with the simple class of 2x2 linear transformations. On the left, we have our input image. After we apply a transformation, we end up with the output image on the right. Each point $\mathbf{p}_1$ (given by x_1, y_1) in the input image is mapped to $\mathbf{p}_2$ (given by x_2, y_2) in the output image. These kinds of transformations can often be described using the 2x2 matrix, T**.** That is, $\mathbf{p}_1$ multiplied by T gives us $\mathbf{p}_2$.

2x2 Linear Transformations



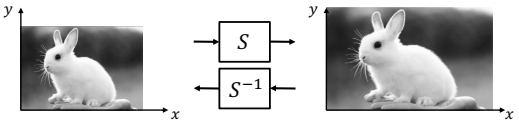
T can be represented by a matrix.

$$\mathbf{p}_2 = T\mathbf{p}_1 \qquad \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = T\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \qquad \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

10

One type of image warping is scaling—stretching or squishing an image. A 2x2 transformation matrix, S, can be formed using the scale factors, a and b, and then applied to each image point. Given that S is invertible, the output image can be warped back to the input image by applying the inverse of S.

Scaling (Stretching or Squishing)



Forward:

$$x_2 = ax_1 \qquad y_2 = by_1$$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = S\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

Inverse:

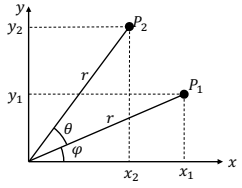
$$x_1 = \frac{1}{a}x_2 \qquad y_1 = \frac{1}{b}y_2$$

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = S^{-1}\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1/a & 0 \\ 0 & 1/b \end{bmatrix}\begin{bmatrix} x_2 \\ y_2 \end{bmatrix}$$

11

Now, for a more complex form of warping, let us consider rotation. For a given point P_1, its coordinates, x_1 and y_1, can be represented using the polar coordinates r and φ. When this point is rotated by an angle θ, we get P_2 with coordinates x_2 and y_2. Using trigonometric identities, we can expand the expressions for x_2 and y_2 and substitute x_1 and y_1 in them.

2D Rotation



$$x_1 = r\cos(\varphi)$$
$$y_1 = r\sin(\varphi)$$

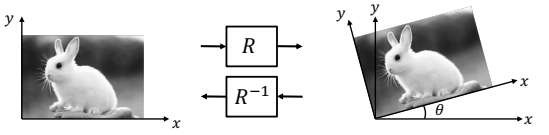
$$x_2 = r\cos(\varphi + \theta)$$
$$x_2 = r\cos\varphi\cos\theta - r\sin\varphi\sin\theta$$
$$x_2 = x_1\cos\theta - y_1\sin\theta$$

$$y_2 = r\sin(\varphi + \theta)$$
$$y_2 = r\cos\varphi\sin\theta + r\sin\varphi\cos\theta$$
$$y_2 = x_1\sin\theta + y_1\cos\theta$$

MATH PRIMER 12

The above equations for x_2 and y_2 can be expressed using a 2x2 matrix denoted by R [1]. To undo the effect of this rotation, we can simply apply the inverse of R to x_2 and y_2.

Rotation



Forward:

$x_2 = x_1 cos\theta - y_1 sin\theta$

$y_2 = x_1 sin\theta + y_1 cos\theta$

$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = R \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$ [1]

Inverse:

$x_1 = x_2 cos\theta + y_2 sin\theta$

$y_1 = -x_2 sin\theta + y_2 cos\theta$

$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = R^{-1} \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}$

13

2x2 transformation matrices can also be used to skew an image, turning it from a rectangular image into a parallelogram. Skews can be applied in any direction, but let us first focus on horizontal skewing, where the amount the *x*-coordinate is modified by is determined by multiplying the *y*-coordinate with a constant. This effectively "pulls" the image into a parallelogram. For skewing in the vertical direction, it is y_2 that is modified by multiplying the *x*-coordinate with a constant, while x_2 remains unchanged.

Skew

Horizontal Skew:

$x_2 = x_1 + m_x y_1$

$y_2 = y_1$

$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = S_x \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 1 & m_x \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$

Vertical Skew:

$x_2 = x_1$

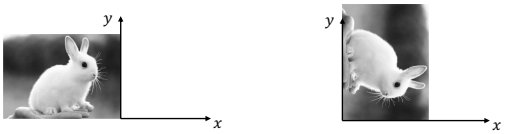
$y_2 = m_y x_1 + y_1$

$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = S_x \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ m_y & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$

14

An image can also be mirrored, or flipped. To flip an image about the *y*-axis, the transformation matrix M_y can be applied to make all the *x*-values negative. To flip across the line $y = x$, we use a matrix M_{xy} that swaps *x* and *y*.

Mirror



Mirror about Y-axis:

$x_2 = -x_1$

$y_2 = y_1$

$M_y = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$

Mirror about line $y = x$:

$x_2 = y_1$

$y_2 = x_1$

$M_{xy} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

15

2x2 transformation matrices have some important properties. First, the origin always maps to the origin, meaning that an input of (0,0) will always produce an output of (0,0). Second, lines always map to lines, meaning that if our input is a line, the output due to the transformation will also be a line. Furthermore, parallel lines will remain parallel. Finally, it is important to note that these transformations are closed under composition. This means that if $\mathbf{p_1}$ is transformed to $\mathbf{p_2}$ with transformation $T_{21,}$ and $\mathbf{p_2}$ is transformed to $\mathbf{p_3}$ with transformation T_{32}, it is possible to compose these transformations to obtain a transformation that maps $\mathbf{p_1}$ to $\mathbf{p_3}$. This transformation T_{31} is equal to the matrix product of the transformations T_{32} and T_{21}.

2x2 Matrix Transformations

Any transformation of the form:

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

- Origin maps to the origin
- Lines map to lines
- Parallel lines remain parallel
- Closed under composition

$$\left.\begin{aligned} \mathbf{p}_2 &= T_{21}\mathbf{p}_1 \\ \mathbf{p}_3 &= T_{32}\mathbf{p}_2 \\ \mathbf{p}_3 &= T_{31}\mathbf{p}_1 \end{aligned}\right\} \quad \mathbf{p}_3 = T_{32}\mathbf{p}_2 = T_{32}T_{21}\mathbf{p}_1 \Rightarrow \quad T_{31} = T_{32}T_{21}$$

16

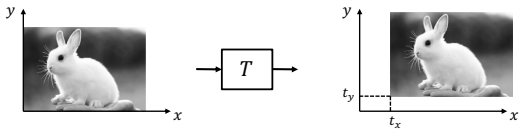
3x3 Image Transformations

Shree K. Nayar
Columbia University

Topic: Image Stitching, Module: Features
First Principles of Computer Vision

17

Translation



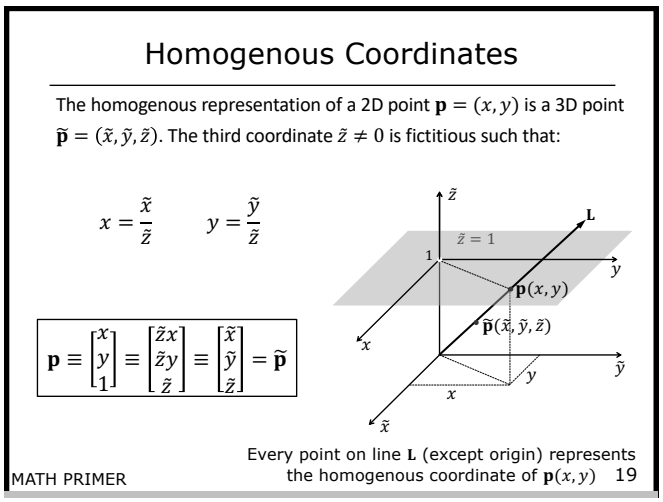
$$x_2 = x_1 + t_x \qquad y_2 = y_1 + t_y$$

Can translation be expressed as a 2x2 matrix? No.

18

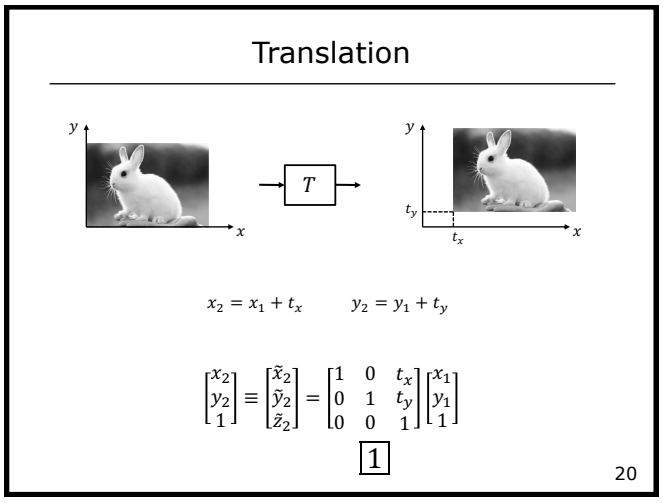
Now, consider the seemingly simple problem of translating (shifting) an image. Given the image on the left, we want to translate it by t_x in the *x*-direction and t_y in the *y*-direction. This can be represented by the two simple expressions shown here. However, there is no way to represent these expressions as a 2x2 transformation matrix. We will address this problem by using homogeneous coordinates.

Homogenous coordinates are a very important concept that is widely used in science and engineering to represent various kinds of transformations as matrices. In the example shown here, the homogeneous representation of the 2D point **p** = (*x,y*) is a 3D point $\tilde{\mathbf{p}} = (\tilde{x}, \tilde{y}, \tilde{z})$. The third coordinate, $\tilde{z}$, is often called a fictitious coordinate and is used to normalize the first two coordinates. To go from $\tilde{\mathbf{p}}$ to **p**, we simply divide both $\tilde{x}$ and $\tilde{y}$ by $\tilde{z}$.



Let's examine this concept from a geometric perspective. Note that **p**(*x*,*y*) is a point in the *x*-*y* plane. To examine what homogeneous representation means, we are going to erect a coordinate frame, $(\tilde{x}, \tilde{y}, \tilde{z})$, such that the *x*-*y* plane lies at $\tilde{z} = 1$. Now, if we consider all the points on the line **L** that goes from the origin through the point **p**, we can say that all these points, except for the origin, are equivalent to one another, and they are equivalent to the point **p**. In other words, every point on line **L,** except the origin, represents the homogeneous coordinates of **p**. That is, if we take any point on **L,** its $\tilde{x}$ and $\tilde{y}$ coordinates can be divided by its $\tilde{z}$ coordinate to get (*x*,*y*).

Returning to the problem of translation, we can see how homogenous coordinates prove useful. Using homogenous coordinates for (x_1, y_1) and (x_2, y_2), we can express translation as a 3x3 transformation matrix $\boxed{1}$, which includes the translation parameters t_x and t_y.



Any of the 2x2 transformations we discussed earlier can also be done using a 3x3 matrix. Furthermore, imagine that we wanted to first skew an image, then translate it, then scale it, and finally rotate it. These transformations do not need to be applied in sequence. Instead, the corresponding transformation matrices can be multiplied, in that sequence, to yield a composition. This composition is a single 3x3 matrix that can be applied to an image to achieve the same result as the sequence of transformations.

Scaling, Rotation, Skew, Translation

$$\begin{bmatrix}\tilde{x}_2\\ \tilde{y}_2\\ \tilde{z}_2\end{bmatrix}=\begin{bmatrix}s_x & 0 & 0\\ 0 & s_y & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x_1\\ y_1\\ 1\end{bmatrix}$$

Scaling

$$\begin{bmatrix}\tilde{x}_2\\ \tilde{y}_2\\ \tilde{z}_2\end{bmatrix}=\begin{bmatrix}1 & m_x & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x_1\\ y_1\\ 1\end{bmatrix}$$

Skew

$$\begin{bmatrix}\tilde{x}_2\\ \tilde{y}_2\\ \tilde{z}_2\end{bmatrix}=\begin{bmatrix}1 & 0 & t_x\\ 0 & 1 & t_y\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x_1\\ y_1\\ 1\end{bmatrix}$$

Translation

$$\begin{bmatrix}\tilde{x}_2\\ \tilde{y}_2\\ \tilde{z}_2\end{bmatrix}=\begin{bmatrix}cos\theta & -sin\theta & 0\\ sin\theta & cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x_1\\ y_1\\ 1\end{bmatrix}$$

Rotation

Composition of these transformations?

21

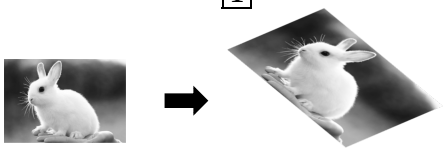
The above class of transformations are categorized as *affine* transformations, for which the general form is shown here $\boxed{1}$. In all cases of the affine transformation, the bottom-most row is always $[0 \quad 0 \quad 1]$. The affine transformation therefore has 6 free parameters, since the 3 values of the bottom-most row are fixed.

Affine Transformation

Any transformation of the form:

$$\begin{bmatrix}x_2\\ y_2\\ 1\end{bmatrix}\equiv\begin{bmatrix}\tilde{x}_2\\ \tilde{y}_2\\ \tilde{z}_2\end{bmatrix}=\begin{bmatrix}a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\tilde{x}_1\\ \tilde{y}_1\\ \tilde{z}_1\end{bmatrix}$$

$\boxed{1}$



22

In the case of affine transformations, the origin does not necessarily map to the origin, since translation might be involved and would shift the origin. However, lines still map to lines, parallel lines remain parallel, and transformations are still closed under composition.

Affine Transformation

Any transformation of the form:

$$\begin{bmatrix}x_2\\ y_2\\ 1\end{bmatrix}\equiv\begin{bmatrix}\tilde{x}_2\\ \tilde{y}_2\\ \tilde{z}_2\end{bmatrix}=\begin{bmatrix}a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\tilde{x}_1\\ \tilde{y}_1\\ \tilde{z}_1\end{bmatrix}$$

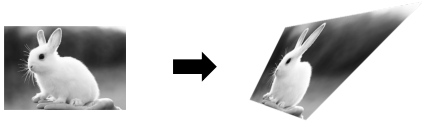
- Origin does not necessarily map to the origin
- Lines map to lines
- Parallel lines remain parallel
- Closed under composition

23

Projective Transformation

Any transformation of the form:

$$\begin{bmatrix}\tilde{x}_2\\ \tilde{y}_2\\ \tilde{z}_2\end{bmatrix} = \begin{bmatrix}h_{11} & h_{12} & h_{13}\\ h_{21} & h_{22} & h_{23}\\ h_{31} & h_{32} & h_{33}\end{bmatrix}\begin{bmatrix}\tilde{x}_1\\ \tilde{y}_1\\ \tilde{z}_1\end{bmatrix} \qquad \tilde{\mathbf{p}}_2 = H\tilde{\mathbf{p}}_1$$

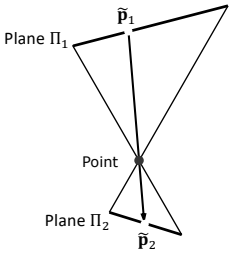


Also called Homography

24

Projective Transformation

Mapping of one plane to another through a point



$$\tilde{\mathbf{p}}_2 = H\tilde{\mathbf{p}}_1$$

$$\begin{bmatrix}\tilde{x}_2\\ \tilde{y}_2\\ \tilde{z}_2\end{bmatrix} = \begin{bmatrix}h_{11} & h_{12} & h_{13}\\ h_{21} & h_{22} & h_{23}\\ h_{31} & h_{32} & h_{33}\end{bmatrix}\begin{bmatrix}\tilde{x}_1\\ \tilde{y}_1\\ \tilde{z}_1\end{bmatrix}$$

Same as imaging a plane through a pinhole

25

When the last row is not restricted to be $[0 \quad 0 \quad 1]$, the transformation matrix is called a projective matrix, also called a homography matrix. A projective matrix maps one plane Π_1 to another plane Π_2 through a point. This is relevant in the context of computer vision because that is exactly how a camera images a plane in the scene—it maps the scene plane to the image plane through an effective center of projection, or pinhole. As we will show, this is important to our application of image stitching because when we take a set of images of a scene by simply rotating the camera about its center of projection, any two images in the set are related via a homography.

Projective Transformation

Homography can only be defined up to a scale.

$$\begin{bmatrix}h_{11} & h_{12} & h_{13}\\ h_{21} & h_{22} & h_{23}\\ h_{31} & h_{32} & h_{33}\end{bmatrix}\begin{bmatrix}\tilde{x}_1\\ \tilde{y}_1\\ \tilde{z}_1\end{bmatrix} \equiv \begin{bmatrix}\tilde{x}_2\\ \tilde{y}_2\\ \tilde{z}_2\end{bmatrix} \equiv k\begin{bmatrix}h_{11} & h_{12} & h_{13}\\ h_{21} & h_{22} & h_{23}\\ h_{31} & h_{32} & h_{33}\end{bmatrix}\begin{bmatrix}\tilde{x}_1\\ \tilde{y}_1\\ \tilde{z}_1\end{bmatrix}$$

If we fix scale such that $\sqrt{\Sigma(h_{ij})^2} = 1$ then 8 free parameters

- Origin does not necessarily map to the origin
- Lines map to lines
- Parallel lines do not necessarily remain parallel
- Closed under composition

26

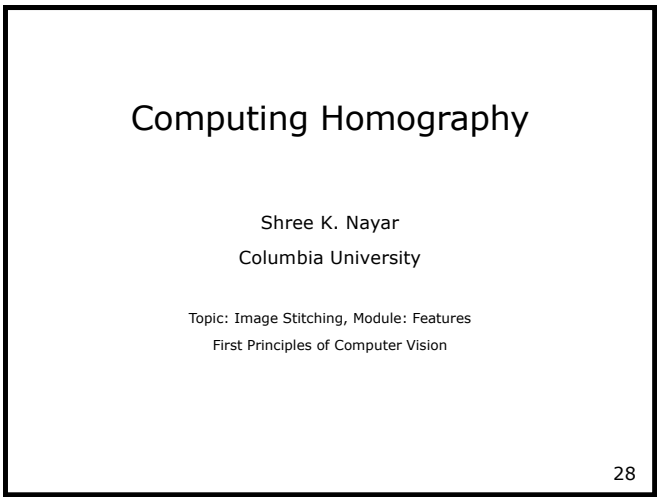
Remember Vanishing Points?

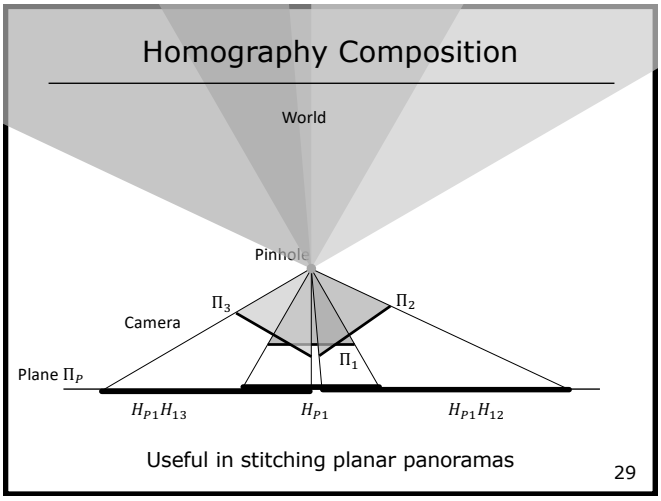
27

An interesting fact about the homography matrix is that it has a scale ambiguity. Since the input and output image coordinates are represented using homogeneous coordinates, you can multiply the homography matrix by any scalar *k* and the output image coordinates x_2 and y_2 will be unaffected. Therefore, the homography matrix is only defined up to an unknown scale. In other words, we can arbitrarily fix the scale of the homography matrix. This can be done in several ways. For instance, we can

set the magnitude of the matrix equal to 1. Irrespective of how the scale is fixed, the end effect is that, although the matrix has 9 elements, it has only 8 free parameters.

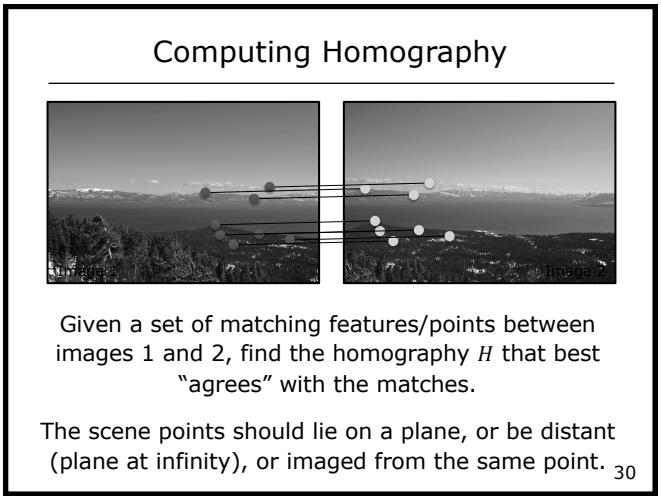
In the case of the homography, the origin does not necessarily map to the origin. Lines still map to lines, which we can show by plugging in an equation of a line for the input and see that the output is a line equation. Transformations are still closed under composition. However, parallel lines do not necessarily remain parallel. Imagine we have a plane in 3D with parallel lines on it, like the railway tracks on the right. We know that in the image of this scene plane, the lines are not necessarily going to remain parallel. It is this effect that results in vanishing points, which we discussed in the lecture on image formation.





As discussed above, the homography is a transformation matrix that maps a point from one plane to another plane through a point of projection. Before we discuss how to compute the homography, let us first examine why this is relevant to the problem of image stitching. In the setting shown here, we have our 3D scene (world) on top and we take a first image in which the camera maps the scene onto plane Π_1. Then, we rotate the camera about its center of projection to take another image that falls on plane Π_2, and then rotate the camera again to take a third image that falls on plane Π_3. If we define another plane, Π_P, we know that there is a homography H_{P1} that relates plane Π_1 to plane Π_P, as they share the same center of projection. So, we can map our image on plane Π_1 to plane Π_P using homography H_{P1}. We can then map our second image, which lies on plane Π_2, to the same plane by mapping plane Π_2 to Π_1 with homography H_{12} and then multiplying H_{12} with H_{P1}. We can do the same thing to the third image as well, which lies on plane Π_3. Therefore, if we have multiple images that share the same center of projection, we can map all the images to a single plane by simply using homographies. The mapped images will appear geometrically consistent with each other (their overlapping fields of view will perfectly align), irrespective of the complexity of the 3D structure of the scene.

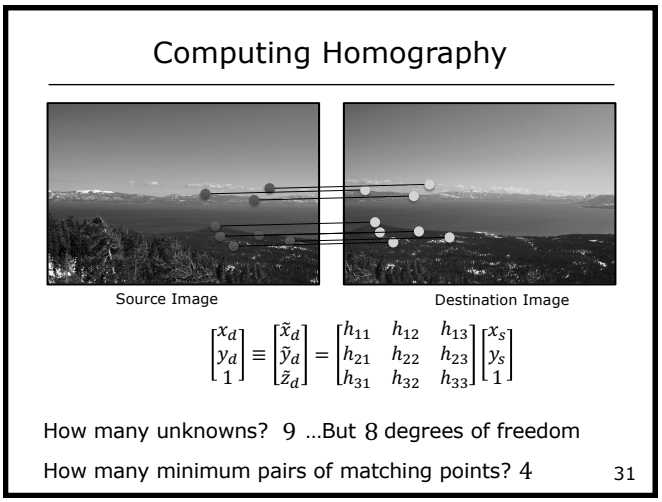
Let us now describe the process of computing the homography between two images. We start by applying the SIFT feature detector to the two images and match features. Given a set of matching features (matching points) between the two images, we want to find the homography that best agrees with the coordinates of the matching points. Once this homography is computed, we can warp one image to the coordinate frame of the other.



It is important to consider when a computed homography is actually valid and hence useful. First, it is always valid if we are capturing images of a 3D scene from the same viewpoint, irrespective of the complexity of the scene. Second, a computed homography is always valid if we are imaging just a plane in the 3D scene, even when the images are taken from different viewpoints. This is because each of the images is related to the plane in the scene via a homography, and hence they are related to each other via a homography (by composition).

In practice, however, images are typically taken from camera viewpoints that are close but not exactly the same, and the scene is not a plane but rather has a complex 3D structure. Even so, if the scene is distant from the camera compared to the distance between camera viewpoints used to capture the images, the scene can be assumed to be a plane at infinity. So, once again, the homography is valid. The case where the homography is not valid is when the scene is close to the camera, has significant depth variations, and the images are taken from different viewpoints.

Now we will show how to compute the homography between two images. We refer to one image as the source and the other as the destination, such that once the homography is computed the source image is warped to the coordinate frame of the destination image. Here, a point $[x_s \quad y_s \quad 1]^T$ in the source image is mapped to a point $[x_d \quad y_d \quad 1]^T$ in the destination image by the homography matrix that we are trying to compute. Again, while there are 9 unknowns in the matrix, we know that the homography can only be computed up to a scale factor and hence we only have 8 degrees of freedom. From the expression shown here we see that each pair of matching points



gives us two equations (one each for x_d and y_d). Therefore, to solve for the 8 degrees of freedom of the homography, we need a minimum of 4 pairs of matching points. In practice, however, we want to use all the matching pairs we have, because that is going to make our estimate of the homography more robust.

Computing Homography

For a given pair i of corresponding points:

$$x_d^{(i)} = \frac{\tilde{x}_d^{(i)}}{\tilde{z}_d^{(i)}} = \frac{h_{11}x_s^{(i)} + h_{12}y_s^{(i)} + h_{13}}{h_{31}x_s^{(i)} + h_{32}y_s^{(i)} + h_{33}} \quad \boxed{1}$$

$$\Rightarrow$$

$$y_d^{(i)} = \frac{\tilde{y}_d^{(i)}}{\tilde{z}_d^{(i)}} = \frac{h_{21}x_s^{(i)} + h_{22}y_s^{(i)} + h_{23}}{h_{31}x_s^{(i)} + h_{32}y_s^{(i)} + h_{33}} \quad \boxed{2}$$

Rearranging the terms:

$$x_d^{(i)}\left(h_{31}x_s^{(i)} + h_{32}y_s^{(i)} + h_{33}\right) = h_{11}x_s^{(i)} + h_{12}y_s^{(i)} + h_{13}$$

$$\Rightarrow$$

$$y_d^{(i)}\left(h_{31}x_s^{(i)} + h_{32}y_s^{(i)} + h_{33}\right) = h_{21}x_s^{(i)} + h_{22}y_s^{(i)} + h_{23}$$

32

Computing Homography

$$x_d^{(i)}\left(h_{31}x_s^{(i)} + h_{32}y_s^{(i)} + h_{33}\right) = h_{11}x_s^{(i)} + h_{12}y_s^{(i)} + h_{13}$$

$$y_d^{(i)}\left(h_{31}x_s^{(i)} + h_{32}y_s^{(i)} + h_{33}\right) = h_{21}x_s^{(i)} + h_{22}y_s^{(i)} + h_{23}$$

Rearranging the terms and writing as linear equation:

$$\begin{bmatrix} x_s^{(i)} & y_s^{(i)} & 1 & 0 & 0 & 0 & -x_d^{(i)}x_s^{(i)} & -x_d^{(i)}y_s^{(i)} & -x_d^{(i)} \\ 0 & 0 & 0 & x_s^{(i)} & y_s^{(i)} & 1 & -y_d^{(i)}x_s^{(i)} & -y_d^{(i)}y_s^{(i)} & -y_d^{(i)} \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

(Known) $\boxed{3}$ $\mathbf{h}$ (Unknown)

33

By expanding the expression in the previous slide, we get the expressions for x_d and y_d, given by $\boxed{1}$ and $\boxed{2}$. We have introduced the superscript *i* here, which corresponds to the index of the matching pair of points. *i* ranges from 1 to N, where N is the total number of detected matching pair. If we multiply through by the denominators in $\boxed{1}$ and $\boxed{2}$, we get the two equations shown at the bottom. We can rewrite these two equations in matrix notation, as shown in $\boxed{3}$. The matrix on the left has two rows because there are two equations, but it includes only *x* terms and *y* terms, which are all known. The vector **h** has all the unknowns, which are the elements of the homography matrix.

We now simply stack up the rows corresponding to the different pairs of matching points to get a large matrix *A*. The result is an overdetermined linear system of equations, $A\,\mathbf{h} = \mathbf{0}$. To solve for **h** we invoke the freedom we have to arbitrarily fix the scale of **h**, by requiring the square of the magnitude of **h** to equal 1.

Computing Homography

Combining the equations for all corresponding points:

$$\begin{bmatrix} x_s^{(1)} & y_s^{(1)} & 1 & 0 & 0 & 0 & -x_d^{(1)}x_s^{(1)} & -x_d^{(1)}y_s^{(1)} & -x_d^{(1)} \\ 0 & 0 & 0 & x_s^{(1)} & y_s^{(1)} & 1 & -y_d^{(1)}x_s^{(1)} & -y_d^{(1)}y_s^{(1)} & -y_d^{(1)} \\ \vdots & & & & & & & & \\ x_s^{(i)} & y_s^{(i)} & 1 & 0 & 0 & 0 & -x_d^{(i)}x_s^{(i)} & -x_d^{(i)}y_s^{(i)} & -x_d^{(i)} \\ 0 & 0 & 0 & x_s^{(i)} & y_s^{(i)} & 1 & -y_d^{(i)}x_s^{(i)} & -y_d^{(i)}y_s^{(i)} & -y_d^{(i)} \\ \vdots & & & & & & & & \\ x_s^{(n)} & y_s^{(n)} & 1 & 0 & 0 & 0 & -x_d^{(n)}x_s^{(n)} & -x_d^{(n)}y_s^{(n)} & -x_d^{(n)} \\ 0 & 0 & 0 & x_s^{(n)} & y_s^{(n)} & 1 & -y_d^{(n)}x_s^{(n)} & -y_d^{(n)}y_s^{(n)} & -y_d^{(n)} \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

A (Known) $\mathbf{h}$ (Unknown)

Solve for $\mathbf{h}$: $\boxed{A\,\mathbf{h} = \mathbf{0}}$ such that $\|\mathbf{h}\|^2 = 1$

34

The computation of the homography can therefore be posed as the well-known constrained least squares problem. Our goal is to find **h** that minimizes the squared magnitude of the product of *A* and **h**, such that the squared magnitude of **h** is equal to one. With some simplifications, we see that our goal is to find the **h** that minimizes $\mathbf{h}^T A^T A \mathbf{h}$, such that $\mathbf{h}^T \mathbf{h} = 1$.

Constrained Least Squares

Solve for $\mathbf{h}$: $\boxed{A\,\mathbf{h} = \mathbf{0}}$ such that $\|\mathbf{h}\|^2 = 1$

Define least squares problem:

$$\min_{\mathbf{h}} \|A\mathbf{h}\|^2 \quad \text{such that } \|\mathbf{h}\|^2 = 1$$

We know that:

$$\|A\mathbf{h}\|^2 = (A\mathbf{h})^T(A\mathbf{h}) = \mathbf{h}^T A^T A \mathbf{h} \quad \text{and} \quad \|\mathbf{h}\|^2 = \mathbf{h}^T\mathbf{h} = 1$$

$$\boxed{\min_{\mathbf{h}} (\mathbf{h}^T A^T A \mathbf{h}) \text{ such that } \mathbf{h}^T\mathbf{h} = 1}$$

MATH PRIMER 35

To solve this, we will define a loss function *L* that combines the constraints from before. We want to find the **h** that minimizes the loss, *L*. To do that, we find the derivative of *L* with respect to **h** and set it equal to zero. This results in a simple expression, which turns out to be the classical eigenvalue problem. If we find the eigenvalues and eigenvectors of A^TA, the **h** we are looking for is the eigenvector that corresponds to the smallest eigenvalue. After we find **h**, we take its elements and rearrange them into the 3x3 homography matrix.

Constrained Least Squares

$$\min_{\mathbf{h}} (\mathbf{h}^T A^T A \mathbf{h}) \text{ such that } \mathbf{h}^T\mathbf{h} = 1$$

Define Loss function $L(\mathbf{h}, \lambda)$:

$$L(\mathbf{h}, \lambda) = \mathbf{h}^T A^T A \mathbf{h} - \lambda(\mathbf{h}^T\mathbf{h} - 1)$$

Taking derivatives of $L(\mathbf{h}, \lambda)$ w.r.t $\mathbf{h}$: $2A^TA\mathbf{h} - 2\lambda\mathbf{h} = \mathbf{0}$

$$\boxed{A^TA\mathbf{h} = \lambda\mathbf{h}} \quad \text{Eigenvalue Problem}$$

Eigenvector $\mathbf{h}$ with smallest eigenvalue λ of matrix A^TA minimizes the loss function $L(\mathbf{h})$.

Matlab: `eig(A'*A)` returns eigenvalues and vectors of A^TA

MATH PRIMER 36

In summary, we apply SIFT to find matching points in the two images, take all the matching points and construct the matrix *A*, find the eigenvector corresponding to the smallest eigenvalue of A^TA, and then rearrange this eigenvector to form the homography matrix.

When computing the homography between two images, all the pairs of matching features were used. However, not all these pairs necessarily correspond to valid matches. For instance, two points in two different images that have the exact same local appearance would be deemed to be a perfect match by the SIFT detector. However, despite their identical appearances, they may not come from the same physical point in the 3D scene. In short, they do not represent a valid match. Unfortunately, there is no way of differentiating between a valid match and an invalid match before computing the homography. We need to therefore make our homography computation resilient to the invalid matches, or outliers.

Dealing with Outliers: RANSAC

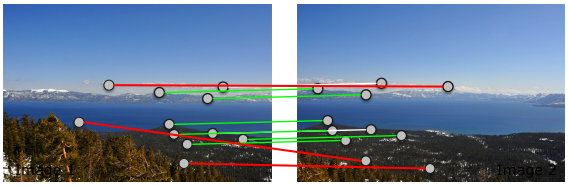
Shree K. Nayar
Columbia University

Topic: Image Stitching, Module: Features
First Principles of Computer Vision

37

Consider these two images where we have several matches, some valid (green) and others invalid (red). We would like to compute a homography that is still valid in the presence of these outliers. This brings us to a technique called RANSAC, which was developed in the early 1980s for computer vision but can be applied to a large class of inlier-outlier problems. Remarkably, RANSAC can work well even when 50% of the data are outliers!

What Could Go Wrong?



Outliers!

We need to robustly compute transformation in the presence of wrong matches.

If number of outliers < 50%, then RANSAC to the rescue!

38

To start with, we are given both a model we wish to fit and data points which include inliers and outliers. Assume that to fit this particular model, we need a minimum of *s* data points. We will randomly choose *s* samples (the minimum number) from our data and then use them to fit the model. We then count the number of points in all our data that fit the model within a certain measure of error, ε. The number of points that fits the model (inliers) is denoted as *M*. We repeat the above process *N* times. The model that yields the largest *M* (inliers) is chosen as the final model. As

RANdom SAmple Consensus

General RANSAC Algorithm:

1. Randomly choose s samples. Typically s is the minimum samples to fit a model.
2. Fit the model to the randomly chosen samples.
3. Count the number M of data points (inliers) that fit the model within a measure of error ε.
4. Repeat Steps 1-3 N times
5. Choose the model that has the largest number M of inliers.

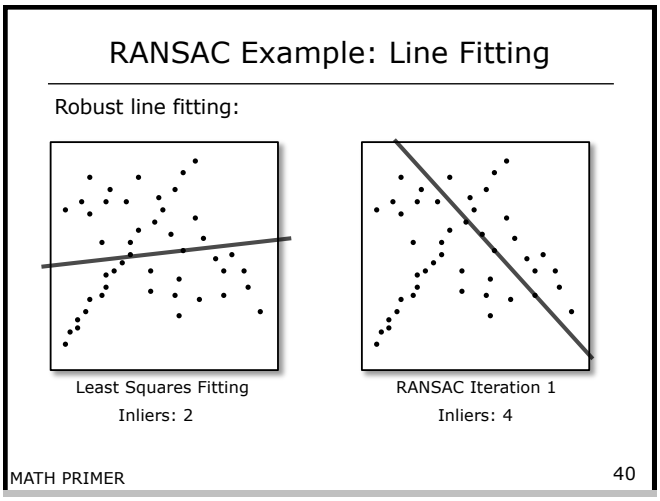
For homography:

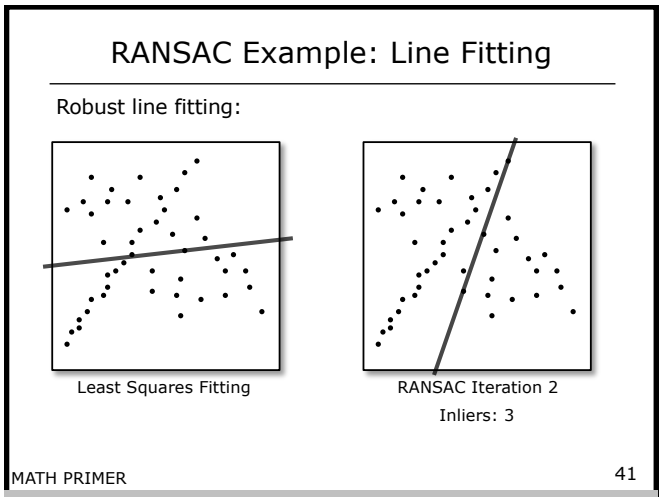
$s = 4$ points. ε is acceptable alignment error in pixels.

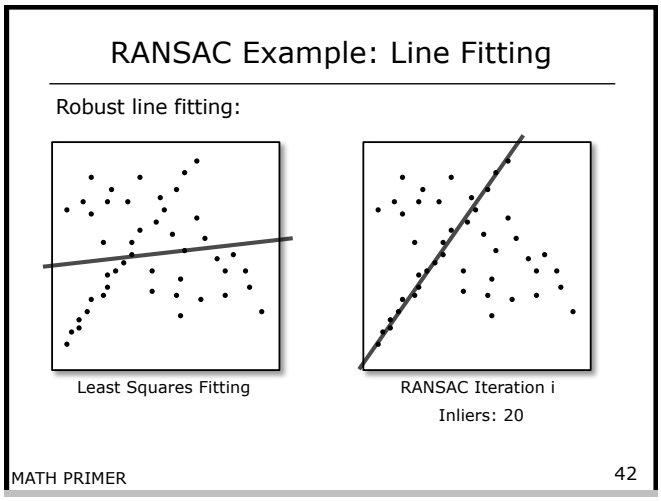
MATH PRIMER [Fischler 1981] 39

a final step, since we know the inliers that correspond to the chosen model, we recompute the model using all the inliers so that it is a more refined model than the one computed using just the minimum number of random samples.

In our case, the model we are trying to fit is the homography matrix and we need a minimum of 4 pairs of matching points to compute the matrix (s =4). Once we have found the homography that maximizes the number M of inliers, we recompute the homography using all M inliers.







To demonstrate how RANSAC works, let us consider the case of line fitting. Shown here is a set of points on a plane. The line we wish to find is seen on the right in slide 42. If we simply fit a line to all the points using the least squares method, we get the line in the left image in slide 40, which is not the line we are looking for. Now, let us take a look at what RANSAC does. Remember that to fit a line we need a minimum of two points. So, we randomly choose two points from the entire set and find the line that passes through them. In the first iteration (the right image in slide 40), we happen to have two more points that lie close to this line, so we have four inliers. In our second iteration (the right image in slide 41), we get a total of three inliers, including the two randomly chosen ones. We keep repeating this process and, sooner or later, we are going to pick two points that lie within the set of points we are looking for. When that happens (right image in slide 42), we get a total of 20 inliers. As a final step, once we have found the line with maximum inliers, we refit the line using just the inliers to get a better estimate of the line.

At this point, we have most of the tools needed to stitch images together to create a panorama. Given a set of images, we begin by selecting a reference image and computing the homography between that image and each of the other images in the set. These homographies can be used to warp all the images into the coordinate frame of the reference image. The images are now aligned and can be merged to create a panorama. However, there are still a few technical issues that need to be addressed to ensure that the final output is of high quality. The first issue is related to warping the images.

Warping and Blending Images

Shree K. Nayar

Columbia University

Topic: Image Stitching, Module: Features

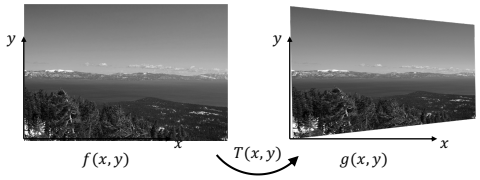
First Principles of Computer Vision

43

Warping Images

Given a transformation T and a image $f(x,y)$, compute the transformed image $g(x,y)$

$$g(x,y) = f(T(x,y))$$

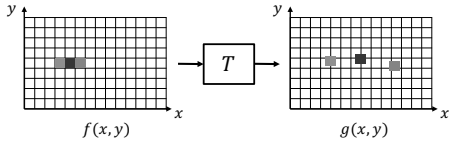


44

Forward Warping

Send each pixel (x,y) in $f(x,y)$ to its corresponding location $T(x,y)$ in $g(x,y)$

$$g(x,y) = f(T(x,y))$$

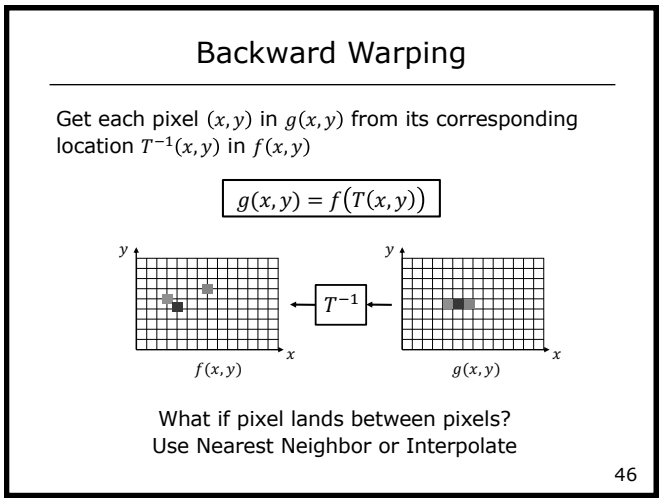


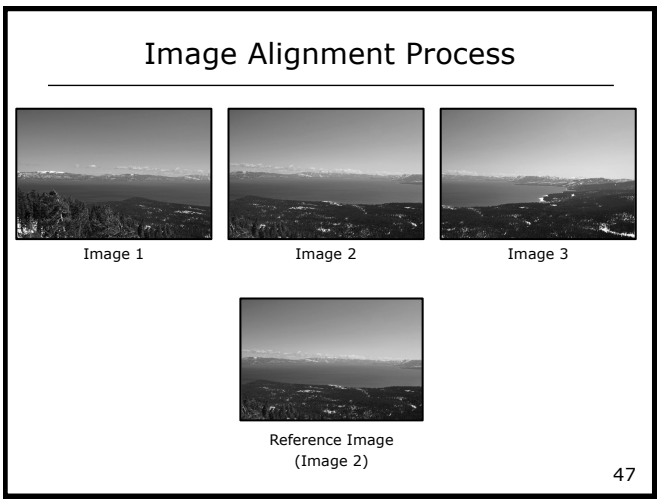
What if pixel lands in between pixels?
What if not all pixels in $g(x,y)$ are filled?
Can result in holes!

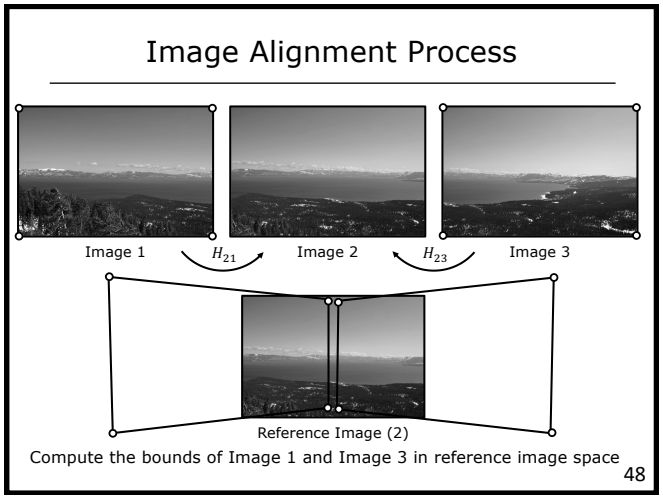
45

Suppose we have an image *f* and we want to apply a geometric transformation *T* to it, to get a transformed image *g*. A naive implementation would be to visit each pixel in the input image *f*, apply the transformation *T* to it, and obtain the corresponding pixel coordinates in output image *g*. Consider how this might is implemented. We have a grid of pixels for *f* and another for *g.* Looking at these two grids, we can see that a transformed pixel may not land exactly at the center of a pixel in *g*. Furthermore, there could be pixels in *g* that do not end up getting filled. That is, this method of warping could result in holes in *g*.

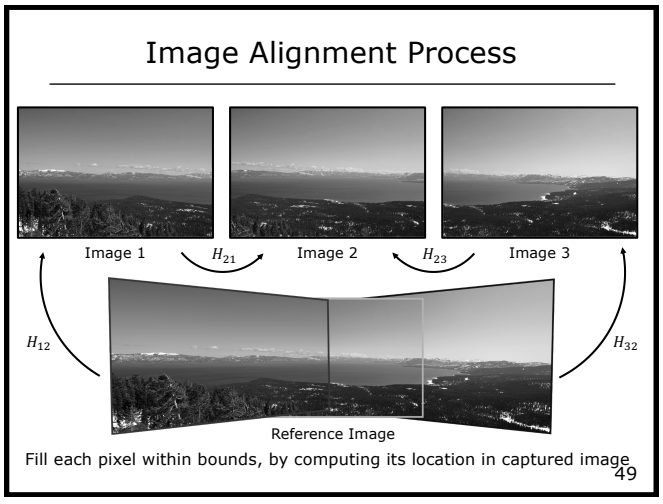
The above issues can be resolved by using backward warping. Forward warping is first applied to the four corners of the input image *f* to get the four corners of the output image *g*. This gives us the bounding box for *g*. Then, we create a grid of pixels for *g* within its bounding box. Our goal is to fill all the pixels in this grid. To do so, we take the coordinates of each pixel in *g* and apply the inverse of the transformation *T* to it. The result is a point that may not lie at the center of a pixel in *f*, in which case, we can use the brightness value of the nearest neighbor. Alternatively, we could use interpolation—take a window of pixels in *f* around the pixel in which the backward warped point lies, and then use all the brightness values within that window to compute the brightness of the pixel. Note that, by using backward warping, we are guaranteed that there will be no holes in *g*.





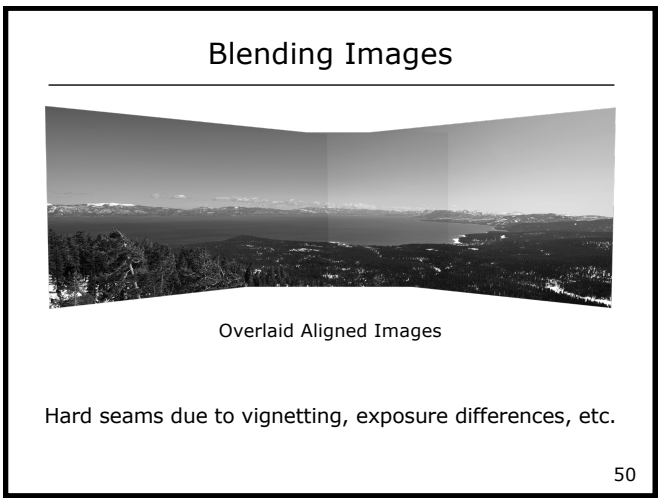


With the backward warping process in place, we are now ready to do image alignment. Let us consider a simple scenario with three input images. Let image two be our reference image, meaning it is the image to which we are going to map the other two images. Since we know the homography H_{21} that relates image one to image two, we take the corners of image one and forward map them to get the bounding box for image one in the reference frame of image two. We do the same for image three. Now, we want to fill in each

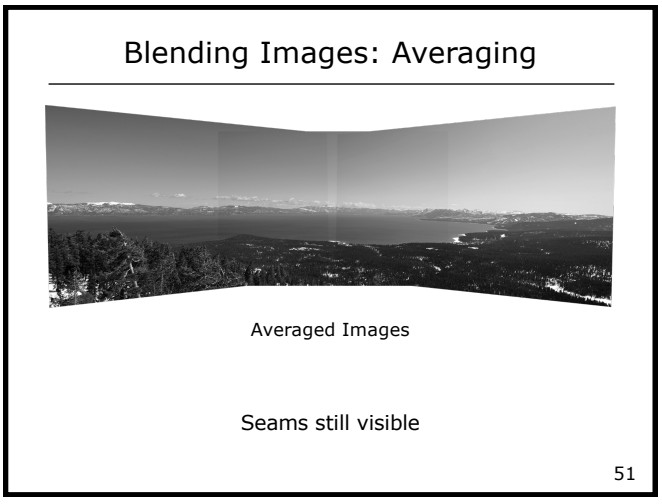


of these bounding boxes. Starting with the bounding box for image one, we take each pixel within it, apply the backward map H_{12}, and pick the brightness value from image one using either the nearest neighbor or interpolation, and write it into the pixel in the bounding box. Once this process is done for all the pixels within the bounding box, image one is warped into the reference frame for image two. The same is done for image three using the backward map H_{32}. The final result is the stack of aligned images shown at the bottom of slide 49.

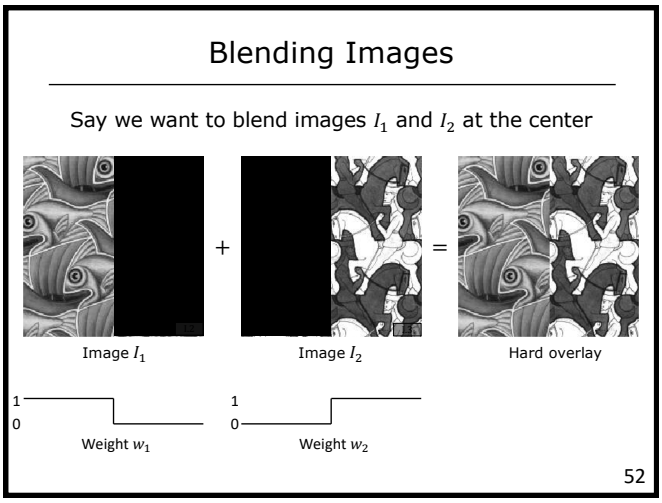
Overlaying the stack of images does produce an image that looks like a panorama, but with clearly visible seams. When capturing the individual images, it is expected that there will be differences in the brightness of the same scene point in the different images. This could be due to differences in the exposures of the images, changes in the scene's lighting while the images are taken, or a spatially varying brightness response of the camera due to effects such as vignetting.



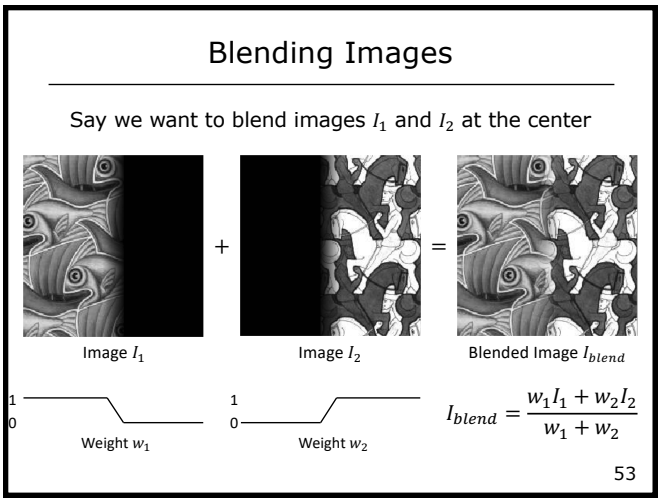
To mitigate these seams, we could try taking the average of all the brightness values (from different images) at each pixel. This does a bit better, but the seams are still visible. The human visual system is extremely sensitive to brightness changes, especially when they lie on smooth contours such as lines.

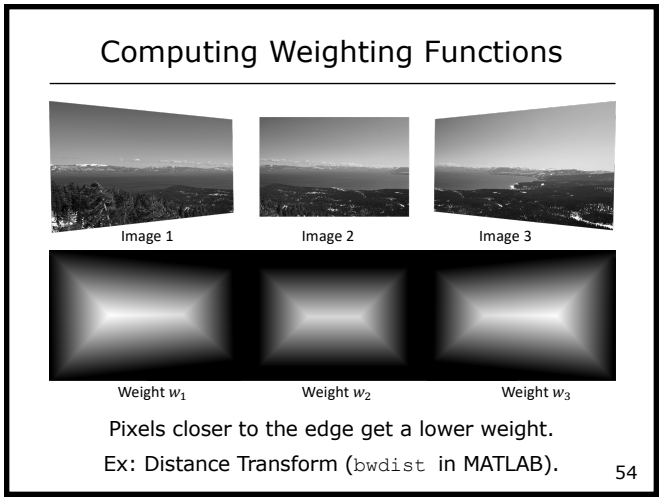


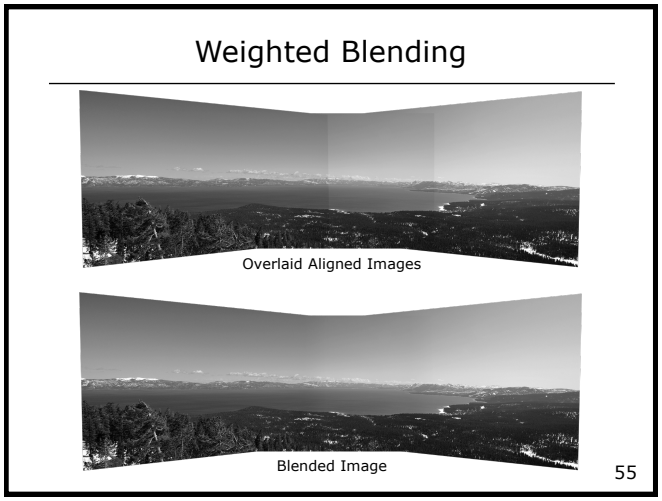
To remove the above seams, we use a method called blending. Let's say that we want to blend the two images I_1 and I_2 shown here, exactly at the center, such that the left half of the output is from I_1 and the right half is from I_2. We can either just cut and paste these halves, or, equivalently, we can use a weighting function that goes from 1 to 0 at the center for I_1 and 0 to 1 at the center for I_2. If you multiply w_1 with I_1 and w_2 with I_2 and sum them together, the result is the image on the right, which has a clearly visible seam.



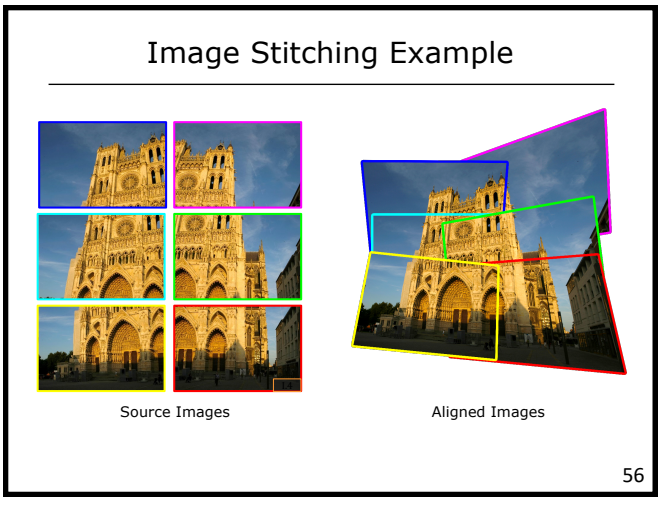
To soften the seam, we can alter the weighting function. For instance, by using a smoother weighting function that gradually goes from a 1 to 0 in the case of I_1 and from 0 to 1 in the case of I_2, we obtain an image like the one shown on the right. The seam is not visible in this case and the image could be perceived as one of a real scene.

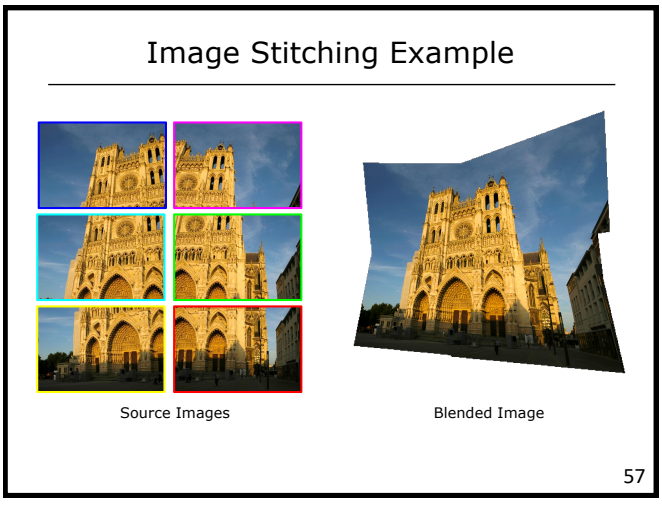






Applying this idea to our stitching problem, we will compute a weighting function for each image, in which the weight of a pixel is a function of the distance of that pixel from the closest boundary point. One way to compute this weighting function is by using the distance transform. Since the weight increases as the distance from the edge increases, the further a pixel is inside an image, the more confidence we have in its brightness during the process of blending. As seen on the right, the result of this simple blending method is a panorama that is seamless.





As a final example, here we have pictures taken of Notre-Dame in Paris. After computing all the homographies, we see the images warped and aligned in slide 56. After blending them, we get the wide-angle image in slide 57.

References and Credits

Shree K. Nayar
Columbia University

Topic: Image Stitching, Module: Features
First Principles of Computer Vision

58

References: Textbooks

Computer Vision: Algorithms and Applications (Chapter 2, 9)
Szeliski, R., Springer

59

References: Papers

[Fischler 1981] Fischler M. A. and Bolles R. C. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography", 1981.

60

Image Credits

I.1 Vanishing Points. Diesel Demon. Licensed under CC BY 2.0.

I.2 https://www.wikiart.org/en/m-c-escher/horseman-1 "Horseman, 1946". © M.C. Escher. Reproduced under WikiArt Fair Use.

I.3 https://www.wikiart.org/en/m-c-escher/fish-boat . "Fish and Boat, 1948". © M.C. Escher. Reproduced under WikiArt Fair Use.

I.4 PTGui. Used with permission.

Acknowledgements: Thanks to Pranav Sukumar and Jenna Everard for their help with transcription, editing and proofreading.

References

[Szeliski 2022] Computer Vision: Algorithms and Applications, Szeliski, R., Springer, 2022.

[Fischler 1981] Fischler M. A. and Bolles R. C. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography", 1981.

[Autopano] Software to make panaromas using SIFT. http://user.cs.tu-berlin.de/~nowozin/autopano-sift/

[Nomura 2007] Y. Nomura, L. Zhang and S.K. Nayar. "Scene Collages and Flexible Camera Arrays." EGSR, 2007.

[Nayar 2022E] Image Processing I, Nayar, S. K., Monograph FPCV-1-4, First Principles of Computer Vision, Columbia University, New York, March 2022.

[Nayar 2022F] Image Processing II, Nayar, S. K., Monograph FPCV-1-5, First Principles of Computer Vision, Columbia University, New York, March 2022.

[Nayar 2022G] Edge Detection, Nayar, S. K., Monograph FPCV-2-1, First Principles of Computer Vision, Columbia University, New York, May 2022.

[Nayar 2022H] Boundary Detection, Nayar, S. K., Monograph FPCV-2-2, First Principles of Computer Vision, Columbia University, New York, June 2022.

[Nayar 2022I] SIFT Detector, Nayar, S. K., Monograph FPCV-2-3, First Principles of Computer Vision, Columbia University, New York, August 2022.

[Nayar 2025H] Camera Calibration, Nayar, S. K., Monograph FPCV-4-1, First Principles of Computer Vision, Columbia University, New York, April 2025.

[Nayar 2025I] Uncalibrated Stereo, Nayar, S. K., Monograph FPCV-4-2, First Principles of Computer Vision, Columbia University, New York, April 2025.