Homography

Let's now look at another image transformation, called a *homography*, which arises very often in real scenes, both manmade and natural. The problems and analysis I will give this lecture are very similar to what we saw last lecture, and we see similar things in lectures to come.

Case 1: scene plane to image pixels

Suppose we have a planar surface in the world (e.g. a wall, a ground plane) and we view it with a camera with projection matrix $\mathbf{P}$. The planar surface is 2D and so we can give it a coordinate system (s,t). Points on the plane are situated in the 3D world and their 3D positions can be expressed in XYZ camera coordinates. We can transform from (s,t) to camera coordinates XYZ by multiplying $(s,t,1)$ by a 4×3 matrix, as follows:

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} a_x & b_x & X_0 \\ a_y & b_y & Y_0 \\ a_z & b_z & Z_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s \\ t \\ 1 \end{bmatrix} \tag{1}$$

The first two columns can be interpreted as direction vectors corresponding to the coordinate system's basis vectors, namely where $(1,0,0)$ and $(0,1,0)$ are mapped to. The third column is the 3D position of the *origin* of the plane, i.e. $(s,t)=(0,0)$. Note that this mapping takes the origin $(s,t)=(0,0)$ to (X_0,Y_0,Z_0); it takes the corner $(s,t)=(0,1)$ to $(X_0+b_x, Y_0+b_y, Z_0+b_z)$, and it takes the corner $(s,t)=(1,0)$ to $(X_0+a_x, Y_0+a_y, Z_0+a_z)$, etc.

The image pixel (x,y) corresponding to a point (s,t) in the scene plane is obtained by

$$\begin{bmatrix} wx \\ wy \\ w \end{bmatrix} = \mathbf{P} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

These two mappings together define a 3×3 matrix $\mathbf{H}$ mapping $(s,t,1)$ to (wx,wy,w),

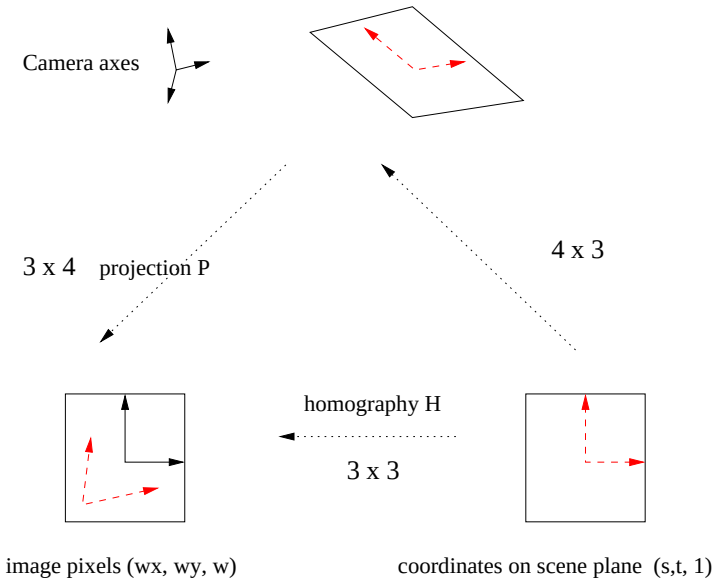
$$\begin{bmatrix} wx \\ wy \\ w \end{bmatrix} = \mathbf{H} \begin{bmatrix} s \\ t \\ 1 \end{bmatrix}.$$

Such an matrix $\mathbf{H}$ is called a *homography*.

Note that the inverse of this mapping is:

$$\begin{bmatrix} w's \\ w't \\ w' \end{bmatrix} = \mathbf{H}^{-1} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

The inverse is well-defined as long as the camera does not lie in the plane defined by Eq. (1) since, in that case, the entire plane would project to a line in the image.



Homography 2 (two cameras, one scene plane)

Suppose the same scene plane is viewed by a second camera (which would have a different XYZ coordinate system). We would now have two homographies $\mathbf{H}_1$ and $\mathbf{H}_2$, defined by the two cameras. This implies that the composite mapping $\mathbf{H}_2\mathbf{H}_1^{-1}$ maps pixels in the first camera to pixels in the second camera. That is, each camera defines a homography of the form

$$\mathbf{H}^{-1} \begin{bmatrix} wx \\ wy \\ w \end{bmatrix} = \begin{bmatrix} s \\ t \\ 1 \end{bmatrix}$$

but the right side is the same for both (since it is independent of the camera), so we just equate the left sides for the two cameras.

Since the product of two 3×3 invertible matrices is itself an invertible matrix, we see that the mapping from pixels of the first camera to pixels of the second camera is a homography. Note: This construction relies critically on the scene being a planar surface.

Homography 3 (panoramas)

Surprisingly, homographies can arise for *general scenes* as well (non-planar), namely if it is viewed by two cameras *from the same center of projection*. In practice this happens when you have one camera and you use it to take more than one image by rotating the camera around the center of projection. This is often done is modern digital cameras when you try to stitch images together to make panorama images.

To see that two images taken under these conditions are related by a homography, let (X, Y, Z) be a scene point visible to camera 1 and written in camera 1's coordinate system.

$$\begin{bmatrix} wx \\ wy \\ w \end{bmatrix} = \mathbf{K} \begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \end{bmatrix}$$

If camera 2's coordinate system is defined by a rotation $\mathbf{R}_{2\leftarrow 1}$ relative to camera 1, we have

$$\begin{bmatrix} \tilde{w}\tilde{x} \\ \tilde{w}\tilde{y} \\ \tilde{w} \end{bmatrix} = \mathbf{K}\mathbf{R}_{2\leftarrow 1} \begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \end{bmatrix}$$

But then

$$\begin{bmatrix} \tilde{w}\tilde{x} \\ \tilde{w}\tilde{y} \\ \tilde{w} \end{bmatrix} = \mathbf{K}\mathbf{R}_{2\leftarrow 1}\mathbf{K}^{-1} \begin{bmatrix} wx \\ wy \\ w \end{bmatrix}$$

But $\mathbf{K}\mathbf{R}_{2\leftarrow 1}\mathbf{K}^{-1}$ is an invertible 3×3 matrix i.e. a homography !

Notice that the distance of the points from the camera plays no role here. This may be somewhat surprising at first glance, but notice that if you are only considering what the scene looks like from a single position then the scene points could be at any distance (even infinity) along the rays that arrive at the camera. Rotating the camera will not have any effect on the directions from which the scene points are seen. So its no really so surprising that the distance to the points plays no role.

[ASIDE: By the way, if you are trying to understand homographies in terms of planes projecting onto planes, then you can think of the last homography as the mapping between the two image projection planes – specifically, where each is defined by pixel coordinates.]

Solving for the homography between two images

Suppose we have N sets of corresponding point pairs (x_i, y_i) and $(\tilde{x}_i, \tilde{y}_i)$ between two images which are *approximately* related by an *unknown* homography $\mathbf{H}$, namely

$$\begin{bmatrix} w\tilde{x}_i \\ w\tilde{y}_i \\ w \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

The points might be image features such as Harris corners. Note that we don't expect to localize these points exactly - there may be errors.

Using N points

We want to solve for $\mathbf{H}$. We can rewrite the equations

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -\tilde{x}_1 x_1 & -\tilde{x}_1 y_1 & -\tilde{x}_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -\tilde{y}_1 x_1 & -\tilde{y}_1 y_1 & -\tilde{y}_1 \\ : & : & : & : & : & : & : & : & : \\ x_N & y_N & 1 & 0 & 0 & 0 & -\tilde{x}_N x_N & -\tilde{x}_N y_N & -\tilde{x}_N \\ 0 & 0 & 0 & x_N & y_N & 1 & -\tilde{y}_N x_N & -\tilde{y}_N y_N & -\tilde{y}_N \end{bmatrix} \begin{bmatrix} H_{11} \\ H_{12} \\ H_{13} \\ H_{21} \\ H_{22} \\ H_{23} \\ H_{31} \\ H_{32} \\ H_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ : \\ 0 \\ 0 \end{bmatrix}$$

But this is just the same type of least squares problem we saw last class. The least squares solution for $\mathbf{H}$ is obtained by taking the $2N \times 9$ data matrix $\mathbf{A}$ and find the eigenvector of $\mathbf{A}^T\mathbf{A}$ having smallest eigenvalue.

[ASIDE: The relationship between this problem and the one we looked at last class is closer than you might think. When we had a calibration grid, we knew a set of points (X_i, Y_i, Z_i). In the case that all points lie on a plane, we can think of the plane as $Z_i = 0$ for all i, and $(s, t) \equiv (X_i, Y_i)$. Since $Z_i = 0$ for all i, it plays no role in the problem and this drops the number of constraints from 12 to 9, namely it drops the third column of the $\mathbf{P}$ matrix.]

Using 4 points only

If we have four corresponding points $(x_i, y_i, \tilde{x}_i, \tilde{y}_i), i = 1, \ldots, 4$ then we get a system of 8 homogenous equations with 9 unknowns. A homography $\mathbf{H}$ that puts the points into *exact* correspondence can be obtained by finding the null space of this system of equation. (That is, we have 9 column vectors each with 8 elements. These column vectors cannot be linearly independent and so there must be a linear combination of them (the H_{ij}) that sums to the zero vector.) We could thus find $\mathbf{H}$ by solving for the null space of the 8×9 matrix $\mathbf{A}$.

(Alternatively, still thinking of this as a least squares problem, we could find the eigenvector of 9×9 matrix $\mathbf{A}^T\mathbf{A}$ with smallest eigenvalue – which we can conclude will be 0, i.e. we know there is an $\mathbf{H}$ in the null space of $\mathbf{A}$ so this is an eigenvector with eigenvalue 0.)

Using RANSAC

In the context of an autonomous vision system such as a robot, you would like to automate these computation, including the problem of finding matching points. This may be difficult to do correctly. Why? In problem 2 where we had two cameras and a scene plane, there may be other surfaces in the scene that do not belong to this plane and that would not be explained by the homography – these other points need somehow to be automatically avoided. Even if you can avoid them, you stil need to correctly match points on the plane as viewed by two different cameras. This might be difficult to do e.g. a bright red flower might be a good feature to detect match from one image to the other, but there might be many bright red flowers.

This should remind you of a problem we saw in lecture 13 where we tried to fit points to lines. So here is the typical RANSAC approach that people take to this problem:

1. Find many feature points (e.g. Harris corners, or SIFT features) in each of the two images: $\{(x_i, y_i)\}$ and $\{(x'_j, y'_j)\}$.

2. For each feature point (x_i, y_i) in one image, find a candidate corresponding feature point in the second image (x'_i, y'_i) whose intensity neighborhood is similar e.g. the SIFT descriptors are similar (recall lecture 12 page 5) or the sum of squared intensity values in a neighbhorhood are similar. This give a set of 4-tuples (x_i, y_i, x'_i, y'_i). You might want to allow each (x_i, y_i) to have several candidate matches (x'_i, y'_i).

3. Randomly choose four 4-tuples and fit an exact homography $\mathbf{H}\,\mathbf{H}_i$ that maps the four $\{(x_i, y_i)\}$ exactly to their corresponding $\{(x'_j, y'_j)\}$. Find the consensus set for that homography, namely find the number of other 4-tuples for whom the *distance* of the 4-tuple from the model is sufficiently small.

The distance could be defined in a variety of ways[1] e.g. compute

$$\begin{bmatrix} w\tilde{x} \\ w\tilde{y} \\ w \end{bmatrix} = \mathbf{H} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

and define

$$dist(\mathbf{H}, x_i, y_i, x_i', y_i') \;=\; \| \, (\tilde{x}, \tilde{y}) - (x', y') \, \|_2 \; .$$

4. After repeating step 3 a certain number of times (or until you find a homography whose consensus set exceeds some pre-determined threshold), choose the homography with the largest consensus set and use that consensus set to re-estimate the homography $\mathbf{H}$ using least squares. The reason you do this final step is that your first homography was chosen to exactly fit four pairs of points, and if there is any noise in those points then the fit will be biased by the particular noise of those points, which is undesirable.

At the end of this lecture, I reviewed the SVD. I am inserting a page break so it is clear to you that this is not tied to homographies. See you on the next page.

[1] Some are better than others – a technical detail that I have decided to prune off.

SVD (Singular Value Decomposition)

Take *any* $m \times n$ matrix $\mathbf{A}$. I want to show how to decompose $\mathbf{A}$ into

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathbf{T}$$

where $\mathbf{U}$ is $m \times m$, $\mathbf{\Sigma}$ is an $m \times n$ matrix that is 0 except for the diagonal elements $\sigma_i = \mathbf{\Sigma}_{ii}$, and $\mathbf{V}$ is $n \times n$. Note if $m \neq n$, then $\mathbf{\Sigma}$ is not square, but we can still talk about the diagonal elements. We will assume that $m \geq n$. (A similar construction can be given when $m < n$.)

The first step is to note that the $n \times n$ matrix $\mathbf{A}^T\mathbf{A}$ is symmetric and positive semi-definite – that's easy to see since $\mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x} \geq 0$ for any real $\mathbf{x}$. So $\mathbf{A}^T\mathbf{A}$ has an orthonormal set of eigenvectors and the eigenvalues are all real and non-negative. Let $\mathbf{V}$ be an $n \times n$ matrix whose columns are the orthonormal eigenvectors of $\mathbf{A}^T\mathbf{A}$. Since the eigenvalues are non-negative, we can write[2] them as σ_i^2.

Define $\mathbf{\Sigma}$ to be an $n \times n$ matrix with values $\Sigma_{ii} = \sigma_i$ on the diagonal and value 0 off the diagonal. (These diagonal elements, σ_i, are called the *singular values* of $\mathbf{A}$. Note that they are the square roots of the eigenvalues of $\mathbf{A}^T\mathbf{A}$.) We now have

$$\mathbf{A}^T\mathbf{A}\mathbf{V} = \mathbf{V}\mathbf{\Sigma}^2.$$

Next define

$$\tilde{\mathbf{U}}_{m\times n} \equiv \mathbf{A}_{m\times n}\mathbf{V}_{n\times n}.$$

Then

$$\tilde{\mathbf{U}}^T\tilde{\mathbf{U}} = \mathbf{V}^\mathbf{T}\mathbf{A}^\mathbf{T}\mathbf{A}\mathbf{V} = \mathbf{\Sigma}^2.$$

Thus, the $\tilde{\mathbf{U}}$ columns are orthogonal and of length σ_i.

We now define $\mathbf{U}$ whose columns are ortho*normal* (length 1), and so

$$\mathbf{U}\mathbf{\Sigma} = \tilde{\mathbf{U}}.$$

Right multiplying both sides by $\mathbf{V}^T$ gives us

$$\mathbf{A}_{m\times n} = \mathbf{U}_{m\times n}\mathbf{\Sigma}_{n\times m}\mathbf{V}^T_{n\times n}.$$

[ASIDE: Typically one defines the *singular value decomposition* of $\mathbf{A}$ to be slightly different than this, namely one defines the $\mathbf{U}$ to be $m \times m$, by just adding $m - n$ orthonormal columns. One also needs to add $m - n$ rows of 0's to Σ to make it $m \times n$, giving

$$\mathbf{A}_{m\times n} = \mathbf{U}_{m\times m}\mathbf{\Sigma}_{m\times n}\mathbf{V}^T_{n\times n}$$

For our purposes there is no important difference between these two decompositions.]

By the way, Matlab has a function `svd` which computes the singular value decomposition. We can use `svd(A)` to compute the eigenvectors and eigenvalues of $\mathbf{A}^T\mathbf{A}$.

[2] Forgive me for using the symbol σ yet again, but σ is *always* used in the SVD.