Camera calibration

To estimate the geometry of 3D scenes from images, we need to know the mapping from 3D scene points to 2D image points. As we saw a few lectures ago, a simple model of this mapping is the camera matrix which is defined by extrinsic and intrinsic parameters. The problem of finding all these parameters is called *camera calibration.*

The camera matrix $\mathbf{P}$ uses homogeneous coordinates. It maps a point $(X, Y, Z, 1)$ in world coordinates to a pixel position (wx, wy, w):

$$\mathbf{P} = \mathbf{KR}[\ \mathbf{I} \mid \ -\mathbf{c}]$$

where $\mathbf{K}$, $\mathbf{R}$, and $\mathbf{I}$ are 3×3 matrices, $\mathbf{c}$ is a 3×1 vector, and so $\mathbf{P}$ is 3×4. $\mathbf{P}$ is also sometimes called a *finite projective camera.*

Estimating P using least squares

How could we estimate $\mathbf{P}$? The standard method assumes we have a set of known points (X_i, Y_i, Z_i) in the scene. For example, we might have a cube of known size and whose faces have a checkerboard pattern on them (also of known size). The corners of the squares on the checkerboard would define 3D points in a world coordinate system defined by one of the corners of the cube and the three edges that meet at this corner.

If we take a photograph of the cube, we could then (by hand) find the pixel coordinates of the known points on the checkerboard, then we would have a set of constraints relating the pixel positions (x_i, y_i) and the scene positions (X_i, Y_i, Z_i). So we would like to find a matrix $\mathbf{P}$ such that:

$$\begin{bmatrix} w_i x_i \\ w_i y_i \\ w_i \end{bmatrix} \approx \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}$$

Note that I have written this as an approximation, since inevitably there will be or measurement error ("noise"). Also note that we have a single $\mathbf{P}$ matrix, and so the w_i coefficients will depend on i.

The trick now is to rid of the w_i as follows:

$$x_i = \frac{w_i x_i}{w_i} \approx \frac{P_{11}X_i + P_{12}Y_i + P_{13}Z_i + P_{14}}{P_{31}X_i + P_{32}Y_i + P_{33}Z_i + P_{34}}$$

$$y_i = \frac{w_i y_i}{w_i} \approx \frac{P_{21}X_i + P_{22}Y_i + P_{23}Z_i + P_{24}}{P_{31}X_i + P_{32}Y_i + P_{33}Z_i + P_{34}}$$

and so

$$x_i(P_{31}X_i + P_{32}Y_i + P_{33}Z_i + P_{34}) \approx P_{11}X_i + P_{12}Y_i + P_{13}Z_i + P_{14}$$
$$y_i(P_{31}X_i + P_{32}Y_i + P_{33}Z_i + P_{34}) \approx P_{21}X_i + P_{22}Y_i + P_{23}Z_i + P_{24}.$$

We can arrange these equations as the product of a $2N \times 12$ matrix and a 12×1 vector:

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -x_1X_1 & -x_1Y_1 & -x_1Z_1 & -x_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -y_1X_1 & -y_1Y_1 & -y_1Z_1 & -y_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ X_N & Y_N & Z_N & 1 & 0 & 0 & 0 & 0 & -x_NX_N & -x_NY_N & -x_NZ_N & -x_N \\ 0 & 0 & 0 & 0 & X_N & Y_N & Z_N & 1 & -y_NX_N & -y_NY_N & -y_NZ_N & -y_N \end{bmatrix} \begin{bmatrix} P_{11} \\ P_{12} \\ P_{13} \\ P_{14} \\ \vdots \\ P_{31} \\ P_{32} \\ P_{33} \\ P_{34} \end{bmatrix} \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The matrix $\mathbf{P}$ has 12 elements, so if we want to solve for $\mathbf{P}$ then we need $N \geq 6$ corresponding point pairs i.e. we need 12 or more data values. In practice the values of $(x_i, y_i, X_i, Y_i, Z_i)$ will contain some measurement error and the projection model will only be an approximation to the actual physical situation. One typically tries to find a least squares solution and uses $N \gg 6$ points.

The matrix $\mathbf{P}$ is undefined up to scale, since it maps homogeneous points to homogeneous points. So we can enforce a solution that $\parallel \mathbf{P} \parallel = 1$, where

$$\parallel \mathbf{P} \parallel = \sqrt{\sum_{ij} P_{ij}^{\,2}}$$

which is called the *Frobenius norm*. Notice that we just have a (version 2) total least squares problem. The matrix $\mathbf{A}$ in the least squares problem is the $2N \times 12$ matrix above, and so $\mathbf{A}^T\mathbf{A}$ is 12×12. We would like the eigenvector $\mathbf{A}^T\mathbf{A}$ with smallest eigenvalue. To get it, we either compute it directly, or we take the SVD of $\mathbf{A}$ and take the column of $\mathbf{V}$ with the smallest singular value.

Data normalization

If you examine the $2N \times 12$ matrix above which we will use as our $\mathbf{A}$ matrix in least squares, we see that some coefficients depend on the image position and some depend on the scene position and some depend on the product of image positions and scene positions. Is this a problem? Yes it is.

The problem here is that image positions (x, y) are given in pixel units and scene positions (X, Y, Z) are given in units such as *meters* or perhaps *mm*, and there is no natural relationship between these two unit systems. In particular, if I use *mm* instead of *m* for scene positions, then the (X, Y, Z) values will be multiplied by 1000. The last column of the matrix does not depend on the scene positions, and so it would have less impact on the solution than columns that do depend on scene position. This would give more weight to the scene positions than the image positions when finding the best fit (namely when determining the eigenvector of $\mathbf{A}^T\mathbf{A}$ with minimum eigenvalue). Similarly, if you use units of kilometers instead of *m* for scene position, then this would have the opposite effect, and now the last column would have more influence. A similar issue arises if we were to arbitrarily translate the scene points by some vector. For example, if we added 10^6 to one or all of the X, Y, Z components, then this would reduce the effect of the last column of the matrix $\mathbf{A}$ which doesn't depend on position.

The way to avoid this problem is to *normalize* the image positions $\{(x_i, y_i)\}$ and scene positions $\{(X_i, Y_i, Z_i)\}$ prior to building the above matrix. This can be done by translating the data points so that their centroid is at the origin, and scaling them so that on average they have unit length. For the image points, the centroid is

$$(\bar{x}, \bar{y}) = \frac{1}{N} \sum_{i=1}^{N} (x_i, y_i)$$

and the average squared distance of the x and y coordinates from the centroid is:

$$\sigma_1^2 = \frac{1}{2N} \sum_{i=1}^{N} (x_i - \bar{x})^2 + (y_i - \bar{y})^2$$

We *normalize* by applying a translation and scaling as follows:

$$(x_i, y_i) \rightarrow (\frac{x_i - \bar{x}}{\sigma_1}, \frac{y_i - \bar{y}}{\sigma_1})$$

Similarly for the scene points, the centroid is

$$(\bar{X}, \bar{Y}, \bar{Z}) = \frac{1}{N} \sum_{i=1}^{N} (X_i, Y_i, Z_i)$$

and the average squared distance of the x and y coordinates from the centroid is:

$$\sigma_2^2 = \frac{1}{3N} \sum_{i=1}^{N} (X_i - \bar{X})^2 + (Y_i - \bar{Y})^2 + (Z_i - \bar{Z})^2$$

and so we normalize by applying a translation and scaling as follows:

$$(X_i, Y_i, Z_i) \rightarrow (\frac{X_i - \bar{X}}{\sigma_2}, \frac{Y_i - \bar{Y}}{\sigma_2}, \frac{Z_i - \bar{Z}}{\sigma_2}).$$

With the normalized values of image and scene position, we can solve for a camera matrix. Let $\mathbf{M_1}$ be the normalization transformation that translates the centroid of the image positions to the origin and scales these translated values by $1/\sigma_1$, and let $\mathbf{M_2}$ be the normalization transformation that translates the centroid of the scene positions to the origin and scales these translated values by $1/\sigma_2$. (See slides for what these matrices look like, but hopefully you can figure that out for yourself.)

$$\begin{bmatrix} \tilde{x}_i \\ \tilde{y}_i \\ 1 \end{bmatrix} = \mathbf{M_1} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \qquad \begin{bmatrix} \tilde{X}_i \\ \tilde{Y}_i \\ \tilde{Z}_i \\ 1 \end{bmatrix} = \mathbf{M_2} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}$$

We then use the normalized data to build our $\mathbf{A}$ matrix for solving the least squares, namely we take the approximation and use the same method as earlier to build an $\mathbf{A}$ matrix.

$$\begin{bmatrix} w_i \tilde{x}_i \\ w_i \tilde{y}_i \\ w_i \end{bmatrix} \approx \mathbf{P}_{normalized} \begin{bmatrix} \tilde{X}_i \\ \tilde{Y}_i \\ \tilde{Z}_i \\ 1 \end{bmatrix}$$

This gives a best fit camera $\mathbf{P}_{normalized}$ for the normalized values. The best estimate for the camera matrix $\mathbf{P}$ for the original (non-normalized) points is:

$$\mathbf{P} \equiv \mathbf{M_1}^{-1}\mathbf{P}_{normalized}\mathbf{M_2}$$

Read this as follows: the camera maps world points to camera pixels. This can be written as mapping world points to normalized world points, then mapping normalized world points to normalized camera points, then mapping normalized camera points to real camera points (undoing the normalization).

Experiments have shown that these normalization transforms are *mandatory* in practice, if one wishes to accurately estimate the matrix $\mathbf{P}$.

ASIDE: Non linear least squares approach

We next turn to a different issue. The least squares error $\parallel \mathbf{Ax} \parallel$ that we use to set up the problem has a simple solution (namely in terms of the eigenvectors of $\mathbf{A^T A}$). However, it is not obvious how to interpret this error in terms of the geometry of the situation.

A more geometrically meaningful error to minimize is

$$\sum_i \{(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2\} \tag{1}$$

where the $(\hat{x}_i, \hat{y}_i)$ are defined by:

$$\begin{bmatrix} \hat{w}_i \hat{x}_i \\ \hat{w}_i \hat{y}_i \\ \hat{w}_i \end{bmatrix} \equiv \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}$$

that is,

$$\hat{x}_i = \frac{P_{11}X_i + P_{12}Y_i + P_{13}Z_i + P_{14}}{P_{31}X_i + P_{32}Y_i + P_{33}Z_i + P_{34}}$$
$$\hat{y}_i = \frac{P_{21}X_i + P_{22}Y_i + P_{23}Z_i + P_{24}}{P_{31}X_i + P_{32}Y_i + P_{33}Z_i + P_{34}}$$

namely $(\hat{x}_i, y_i)$ are the predicted image positions given a given matrix $\mathbf{P}$ and measured 3D scene point positions (X_i, Y_i, Z_i). This error is the average pixel distance between the positions predicted by the model $\mathbf{P}$ and the positions (x_i, y_i) measured in the image. Note that this is a different error than before. (It is similar, but not exactly the same.)

This new error measure can be minimized using a *non-linear* least squares method. The basic idea is to approximate this new error measure by taking a first order Taylor series approximation of the error function in (1) with respect to the variables P_{ij}. This would yield a regression problem (version 1 least squares) which one could solve exactly. One could then take this solution and iterate to find a better solution, similar to what was done in the Lucas-Kanade method. Details omitted.

Factoring P into internal and external camera parameters

Regardless of which method one uses to estimate $\mathbf{P}$, one then needs to decompose $\mathbf{P}$ into the product of a 3×3 upper triangular matrix $\mathbf{K}$ and a 3×4 matrix $\mathbf{R}[\mathbf{I}| - \mathbf{c}]$. This can be done using the *RQ decomposition*, which is a technique from linear algebra. In the RQ decomposition, R refers to a right upper triangular matrix (everything below the diagonal is zero) and Q refers to an orthonormal matrix. FYI, Matlab doesn't implement the RQ decomposition. It only implements the QR decomposition, which is more standard. I have added the details in an Appendix below.

In the remainder of this lecture, I introduced homographies. The lecture notes for this part can be found in lecture 17.

Appendix: Camera Calibration

You will not be examined on this Appendix, but the method is nice so I encourage you to read through it to get the idea.

Step 1: normalize P

We note that the elements $(P_{31} P_{32} P_{33})$ are not all zero. (Recall from the Exercises that this 3-vector is normal to the projection plane. It cannot be all 0's if we indeed have a projection matrix.) The first step is to divide each element of $\mathbf{P}$ by $\parallel (P_{31}, P_{32}, P_{33}) \parallel$ so that this vector is a unit normal.

For the next steps, we consider only the left 3×3 submatrix of $\mathbf{P}$ whose 3rd row is now of length 1. Let $\tilde{\mathbf{P}}$ be the 3×3 matrix which is the first three columns of $\mathbf{P}$. We decompose $\tilde{\mathbf{P}}$ into $\mathbf{KR}$. To to this, we want to find an orthonormal matrix $\mathbf{R}$ such that $\tilde{\mathbf{P}}\mathbf{R}^T = \mathbf{K}$.

Step 2: find R

We will construct a rotation matrix $\mathbf{R}$ in three steps. First, we find a rotation matrix $\mathbf{R}_{Z,\theta}$ by angle θ about the Z axis, such that $\tilde{\mathbf{P}}\mathbf{R}_{Z,\theta}$ has its $(3,1)$ element equal to zero. Define

$$\mathbf{R}_{Z,\theta} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

and choose θ such that

$$P_{31}\cos\theta - P_{32}\sin\theta = 0$$

and so we need

$$\theta = \arctan(P_{31}/P_{32}).$$

Note that a Z-rotation does not affect the 3rd column of $\tilde{\mathbf{P}}$.

Second, we define a rotation matrix about the X axis (which doesn't change the X value)

$$\mathbf{R}_{X,\beta} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\beta & \sin\beta \\ 0 & -\sin\beta & \cos\beta \end{bmatrix}$$

such that $\mathbf{P}\mathbf{R}_{z,\theta}\mathbf{R}_{X,\beta}$ has its (3,1) and (3,2) elements equal to 0. Using the same method as above, take

$$\beta = \arctan\ \left(\frac{(\tilde{\mathbf{P}}\mathbf{R}_{Z,\theta})_{3,2}}{(\tilde{\mathbf{P}}\mathbf{R}_{Z,\theta})_{33}} \right).$$

The last row of $\tilde{\mathbf{P}}\mathbf{R}_{z,\theta}\mathbf{R}_{X,\beta}$ must now be be $(0, 0, \pm 1)$. Also, recall that we began by normalizing such that the last row of $\tilde{\mathbf{P}}$ was of unit length. The two rotations above will not change this property since rotations preserve length.

Third, we find a rotation matrix $\mathbf{R}_{Z,\gamma}$ which sets element $(2, 1)$ to 0. Again this is done by a suitable choice of rotation angle, namely

$$\gamma = \arctan\ \left(\frac{(\tilde{\mathbf{P}}\mathbf{R}_{Z,\theta}\mathbf{R}_{X,\beta})_{2,1}}{(\tilde{\mathbf{P}}\mathbf{R}_{Z,\theta}\mathbf{R}_{X,\beta})_{2,2}} \right).$$

Note that this X rotation doesn't affect the third row since its first two elements are 0.

We now have that $\tilde{\mathbf{P}}\mathbf{R}_{Z,\theta}\ \mathbf{R}_{X,\beta}\ \mathbf{R}_{Z,\gamma}$. is an upper triangular 3×3 matrix. Since the 3rd row this matrix has length 1, and since elements $(3, 1)$ and $(3, 2)$ are 0, element $(3, 3)$ must be either 1 or -1.

Finally, we would like to transform this into the $\mathbf{K}$ matrix, and so we need the diagonal elements (1,1), (2,2) and (3,3) to be non-negative. If any of these diagonal elements of $\tilde{\mathbf{P}}\mathbf{R}_{Z,\theta}\ \mathbf{R}_{X,\beta}\ \mathbf{R}_{Z,\gamma}$ are negative, then we need to reflect about the X, Y, Z axis, and so we have

$$\mathbf{K} = \tilde{\mathbf{P}}\mathbf{R}_{Z,\theta}\ \mathbf{R}_{X,\beta}\ \mathbf{R}_{Z,\gamma} \begin{bmatrix} \pm 1 & 0 & 0 \\ 0 & \pm 1 & 0 \\ 0 & 0 & \pm 1 \end{bmatrix}.$$

This gives us $\mathbf{K} = \tilde{\mathbf{P}}\mathbf{Q}$ where $\mathbf{Q}$ is orthonormal, and so $\tilde{\mathbf{P}} = \mathbf{K}\mathbf{Q}^T = \mathbf{K}\mathbf{R}$.

Step 3: obtain camera position c

Since **K R (-c)** is the fourth column of **P**, we can obtain **-c** by multiplying

$$\mathbf{R}^T\mathbf{K}^{-1} \begin{bmatrix} P_{14} \\ P_{24} \\ P_{34} \end{bmatrix} = -\mathbf{c}$$

i.e. $\mathbf{R}^T$ is a rotation matrix, so $\mathbf{R}^T = \mathbf{R}^{-1}$.