Reconstruction - I

September 1, 2017

A camera converts the 3D world into a 2D image. One of the goals in computer vision is to reverse this process. That is, from one or more images, we want to reproduce the 3D structure of the world. By 3D structure here, I mean both the 3D structure of the scene, as well as its location and orientation relative to the camera (or alternatively, the camera's pose).

As we have seen, the image formation process collapses an entire ray into a single point. This means that we lose information in the image formation process. Therefore to reverse this process, we need to either make assumptions, or assume the presence of additional information.

A lot of the classical work in this is aimed at doing robust (to outliers, missing information etc) reconstruction from minimal additional information, and uses the geometry of image formation extensively. Here we'll look at a few subproblems that have been addressed:

1. Given a few 3D points and the pixels they project to, what are the parameters of the camera? Here we know at least some of the 3D structure of the world, and we want to know how it is oriented with respect to the camera. This problem is typically called *camera calibration* (if we are interested in the camera) or *pose estimation* (if we are talking of the pose of the scene relative to the camera).

2. Given multiple images from cameras with known parameters (i.e., their relationships to each other are known), and given *correspondences* between the pixels in the images, what is the 3D structure of the scene? In this case we will see that we can get the 3D locations of points in the world by *triangulation*.

3. Given multiple images from cameras with *unknown parameters*, but given *correspondences* between pixels in the images, how do we recover *both* the camera parameters and the 3D structure of the scene? This problem is called the *structure from motion* problem.

Let us look at how each of these problems can be solved.

1 Camera calibration

Suppose we have a set of 3D points $\mathbf{P}_i$ for which we know the corresponding 2D image locations $\mathbf{p}_i = (x_i, y_i)$. Let us write the 3D points in homogenous coordinates $\vec{\mathbf{P}}_i = (X_i, Y_i, Z_i, 1)$. We can write the unknown camera projection matrix as $\mathbf{M}$, where $\mathbf{M} = \mathbf{K}[\mathbf{R}|\mathbf{t}]$ for unknown $\mathbf{K}, \mathbf{R}, \mathbf{t}$. Then, we have that:

$$\vec{\mathbf{p}}_i \sim \mathbf{M}\vec{\mathbf{P}}_i \tag{1}$$

Note that $\mathbf{M}$ and $\lambda\mathbf{M}$ will lead to the same image. Intuitively, multiplying $\mathbf{M}$ by $\lambda > 1$ corresponds to moving the image plane further away while at the same time stretching the coordinate system of the image. Thus we need to either fix an entry in $\mathbf{M}$ to be 1, or we can look for the solution with unit norm etc.

In terms of the inhomogenous coordinates x_i and y_i:

$$x_i = \frac{\mathbf{M}_1^T\vec{\mathbf{P}}_i}{\mathbf{M}_3^T\vec{\mathbf{P}}_i} = \frac{M_{11}X_i + M_{12}Y_i + M_{13}Z_i + M_{14}}{M_{31}X_i + M_{32}Y_i + M_{33}Z_i + M_{34}} \tag{2}$$

$$y_i = \frac{\mathbf{M}_2^T\vec{\mathbf{P}}_i}{\mathbf{M}_3^T\vec{\mathbf{P}}_i} = \frac{M_{21}X_i + M_{22}Y_i + M_{23}Z_i + M_{14}}{M_{31}X_i + M_{32}Y_i + M_{33}Z_i + M_{34}} \tag{3}$$

$$\tag{4}$$

where $\mathbf{M}_i$ is the i-th row of $\mathbf{M}$, that is, $\mathbf{M} = \begin{pmatrix} \mathbf{M}_1^T \\ \mathbf{M}_2^T \\ \mathbf{M}_3^T \end{pmatrix}$.

How many corresponding points do we need? $\mathbf{M}$ has 11 unknowns (3×4 matrix, but minus 1 since scale factor is ambiguous). Each correspondence gives us 2 equations. So we need a minimum of 6 points.

However, not all of these 6 points should lie on a plane. To see this, we need to look at the equation of a plane. In non-homogenous coordinates, the equation of a plane is:

$$N_xX + N_yY + N_zZ + d = 0 \tag{5}$$

In homogenous coordinates, this can be written as $\mathbf{N}^T\vec{\mathbf{P}} = 0$, where $\mathbf{N} = [N_x, N_y, N_z, d]^T$. Thus, if a point $\vec{\mathbf{P}}$ lies on this plane, then taking the dot product of this with $\mathbf{M}_i$ will zero out any component along $\mathbf{N}$. So if all points lie on this plane, we will not have any information about the component of $\mathbf{M}_i$ along $\mathbf{N}$.

Since there are three row vectors we are trying to estimate, and each point gives us two equations, we need at least *two points not in the same plane* to estimate $\mathbf{M}$.

We can solve these equations in two ways. We can convert these equations into linear equations in $\mathbf{M}$ by multiplying out the denominator, and then solve it in a least squares sense by minimizing:

$$\sum_i (\mathbf{M}_3^T\vec{\mathbf{P}}_i x_i - \mathbf{M}_1^T\vec{\mathbf{P}}_i)^2 + (\mathbf{M}_3^T\vec{\mathbf{P}}_i y_i - \mathbf{M}_2^T\vec{\mathbf{P}}_i)^2 \tag{6}$$

Or we can directly minimize the *reprojection error* using non-linear minimization techniques:

$$\sum_i (x_i - \frac{\mathbf{M}_1^T \vec{\mathbf{P}}_i}{\mathbf{M}_3^T \vec{\mathbf{P}}_i})^2 + (y_i - \frac{\mathbf{M}_2^T \vec{\mathbf{P}}_i}{\mathbf{M}_3^T \vec{\mathbf{P}}_i})^2 \tag{7}$$

The latter objective is a physically meaningful objective: it measures the squared distance between the true projection of each point, and the projection based on the estimated camera projection M.

How do we get the camera parameters $\mathbf{K}, \mathbf{R}, \mathbf{t}$ from $\mathbf{M}$? The first 3×3 submatrix of $\mathbf{M}$ is $\mathbf{KR}$. Here $\mathbf{K}$ is known to be upper triangular, and $\mathbf{R}$ is known to be a rotation. This kind of decomposition is called an RQ decomposition in linear algebra and is fairly standard. Once we have $\mathbf{K}$, we can get $\mathbf{t}$ from the last column of $\mathbf{M}$.

A typical way camera calibration is done is by placing a *calibration gig* (a known checkerboard pattern placed at a specific location in the scene). If we have multiple fixed cameras, e.g., on a self-driving car, then we can place this calibration gig at a known location in the scene, and calibrate all the cameras. Then, once we have calibrated cameras, we can reconstruct any scene from the images using these cameras using *triangulation*.

2 Triangulation

What does camera calibration give us? It tells us how the camera coordinate system is related to the world. Thus given a pixel in the camera, which corresponds to a ray in the camera coordinate system, we can know what ray this ray corresponds to in the world coordinate system.

However, this information alone is not enough to tell us where along the ray the 3D point lies. But if we have *multiple* calibrated cameras, and if we knew *correspondences*, i.e, for a pixel in one camera, if we knew the corresponding pixel in all the others, we can compute the corresponding rays and *intersect* them to identify the true location of the 3D point.

Mathematically, it is simpler and easier to minimize the reprojection error. Concretely, suppose $(x^{(j)}, y^{(j)})$ are the projections of a 3D point $\mathbf{P} = (X, Y, Z)$ in the j-th camera. Suppose the projection matrix of the j-th camera is $\mathbf{M}^{(j)} = \mathbf{K}^{(j)}[\mathbf{R}^{(j)}|\mathbf{t}^{(j)}]$ Then, following Equation (7), the reprojection error is:

$$\sum_j (x^{(j)} - \frac{\mathbf{M}_1^{(j)T} \vec{\mathbf{P}}}{\mathbf{M}_3^{(j)T} \vec{\mathbf{P}}})^2 + (y^{(j)} - \frac{\mathbf{M}_2^{(j)T} \vec{\mathbf{P}}}{\mathbf{M}_3^{(j)T} \vec{\mathbf{P}}})^2 \tag{8}$$

3 Structure-from-motion

Now suppose we have no known 3D points to calibrate the cameras. All we have are a set of images $j = 1, \ldots n$, and a set of corresponding image points in these images, with $\vec{\mathbf{p}}_i^{(j)}$ being the image of the i-th point in the j-th image.

Then there are two sets of unknowns: The true 3D locations of these points $\vec{\mathbf{P}}_i$, and the camera projection matrices $\mathbf{M}^{(j)}$. One way to solve for these unknowns is to minimize the reprojection error as before:

$$\sum_{ij}(x_i^{(j)} - \frac{\mathbf{M}_1^{(j)T}\vec{\mathbf{P}}_i}{\mathbf{M}_3^{(j)T}\vec{\mathbf{P}}_i})^2 + (y_i^{(j)} - \frac{\mathbf{M}_2^{(j)T}\vec{\mathbf{P}}_i}{\mathbf{M}_3^{(j)T}\vec{\mathbf{P}}_i})^2 \tag{9}$$

Note that often we may have thousands of cameras and millions of 3D points, so this objective has billions of terms. However, we can take advantage of the problem structure: note that if the camera projection matrices $\mathbf{M}^{(j)}$ are fixed, each point can be optimized independently, and if the 3D points are fixed, each camera matrix can be optimized independently. "Bundle adjustment" is the broad technique used to do such minimization.

However, apart from just minimization of reprojection error, there is a lot of geometric structure in the problem which shows up as useful invariants, which we describe next.

4 Epipolar geometry

Recall the equations that govern how a point $\mathbf{P}$ in the world project onto an image:

$$\vec{\mathbf{p}} \sim \mathbf{K}[\mathbf{R}|\mathbf{t}]\vec{\mathbf{P}} \tag{10}$$

When we have two cameras, then there are two such equations at play. Let $\vec{\mathbf{p}}_1$ and $\vec{\mathbf{p}}_2$ be the corresponding image locations of a world point $\vec{\mathbf{P}}$. Let us assume for the moment that the matrix $\mathbf{K}$, which governs the units and coordinate system on the image, is $\mathbf{I}$ for both cameras. Alternatively, if we know $\mathbf{K}$ beforehand, we can pre-multiply both sides of the equation by $\mathbf{K}^{-1}$. Without loss of generality, we can choose the coordinate system of the world to be the coordinate system of the first camera. Thus we have the following two equations:

$$\vec{\mathbf{p}}_1 \sim [\mathbf{I}|\mathbf{0}]\vec{\mathbf{P}} \tag{11}$$

$$\vec{\mathbf{p}}_2 \sim [\mathbf{R}|\mathbf{t}]\vec{\mathbf{P}} \tag{12}$$

Now, we can write $\vec{\mathbf{P}}$ in terms of the non-homogenous coordinates $\mathbf{P}$ as $\vec{\mathbf{P}} \sim \begin{pmatrix} \mathbf{P} \\ 1 \end{pmatrix}$. We can also replace equivalence in the above two equations by introducing an unknown scale factor:

$$\lambda_1\vec{\mathbf{p}}_1 = \mathbf{P} \tag{13}$$

$$\lambda_2\vec{\mathbf{p}}_2 = \mathbf{R}\mathbf{P} + \mathbf{t} \tag{14}$$

Using the first equation in the second:

$$\lambda_2\vec{\mathbf{p}}_2 = \lambda_1\mathbf{R}\vec{\mathbf{p}}_1 + \mathbf{t} \tag{15}$$

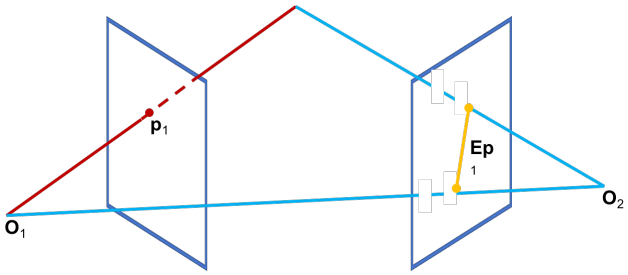

Figure 1: Epipolar lines. In orange is the epipolar line corresponding to point $\mathbf{p}_1$.

.

Take a cross product of $\mathbf{t}$ with equation 17.

$$\lambda_2 \mathbf{t} \times \vec{\mathbf{p}}^2 = \lambda_1 \mathbf{t} \times \mathbf{R}\vec{\mathbf{p}}^1 \tag{16}$$

Now take a dot product with $\vec{\mathbf{p}}^2$:

$$0 = \lambda_1 \vec{\mathbf{p}}_2 \cdot (\mathbf{t} \times \mathbf{R}\vec{\mathbf{p}}_1) \tag{17}$$

We can write this equation in matrix form. To do so, note that the cross product $\mathbf{t} \times \mathbf{x} = \hat{\mathbf{t}}_\times \mathbf{x}$, where

$$\hat{\mathbf{t}}_\times = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} \tag{18}$$

Using this, Equation 17 becomes:

$$\vec{\mathbf{p}}_2^T \hat{\mathbf{t}}_\times \mathbf{R}\vec{\mathbf{p}}_1 = 0 \tag{19}$$

The matrix $\mathbf{E} = \hat{\mathbf{t}}_\times \mathbf{R}$ is called the *essential matrix*.

Note that for a fixed $\vec{\mathbf{p}}_1$, this is the equation of a line: $\mathbf{l}^T \vec{\mathbf{p}}_2 = 0$, where $\mathbf{l} = \vec{\mathbf{p}}_1^T \mathbf{E}^T$. Similarly, for a fixed $\vec{\mathbf{p}}_2$, this is a line in $\vec{\mathbf{p}}_1$. These lines are called *epipolar lines* (Figure 1). An epipolar line is actually the image in one camera of the ray that corresponds to an image pixel in the other camera.

Note that Equation (19) is a single linear equation in the entries of $\mathbf{E}$. $\mathbf{E}$ has $3 \times 3 = 9$ parameters, minus 1 for scale, so 8 free parameters. Thus, we can find $\mathbf{E}$ with 8 pairs of corresponding points.

After we find out $\mathbf{E}$, we can find $\mathbf{R}$ and $\mathbf{t}$ as follows. Observe that $\mathbf{t}^T \mathbf{E} = \mathbf{t}^T \hat{\mathbf{t}}_\times \mathbf{R} = 0$. Thus, $\mathbf{t}$ can be obtained as the *left singular vector* of $\mathbf{E}$ with 0 singular value. Once we know $\mathbf{t}$, we can obtain $\mathbf{R}$ by relating the SVD of $\hat{\mathbf{t}}_\times$ and $\mathbf{E} = \hat{\mathbf{t}}_\times \mathbf{R}$. If $\hat{\mathbf{t}}_\times = \mathbf{U}\mathbf{S}\mathbf{V}^T$, then $\mathbf{E} = \mathbf{U}'\mathbf{S}'\mathbf{V}'^T$. Thus:

$$\mathbf{U}'\mathbf{S}'\mathbf{V}'^T = \mathbf{E} = \hat{\mathbf{t}}_\times \mathbf{R} = \mathbf{U}\mathbf{S}\mathbf{V}^T\mathbf{R} \tag{20}$$

suggesting that $\mathbf{U} = \mathbf{U}'$, $\mathbf{S} = \mathbf{S}'$ and $\mathbf{V}^T\mathbf{R} = \mathbf{V}'^T$. The last equation gives us $\mathbf{R}$. However, there are a few sign ambiguities in these equations since SVD is unique upto a sign. We need to pick the most sensible solution from the family of solutions.