Structure-from-motion: factorization method

Last lecture we looked at a structure from motion method that used two adjacent frames in a video. It was assumed that the instantaneous motion vectors could be computed reliably at many points, especially points near depth discontinuities. It was also assumed that the field of view was large so that the camera rotation field was not parallel i.e. second order terms were significant. Finally, it was also assumed that perspective effects were significant, namely there were large variations in depths; specifically, large differences in inverse depths at depth discontinuities which gave parallax information.

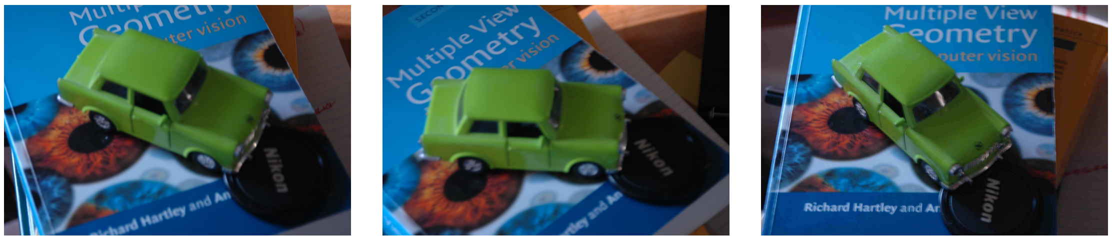
The above conditions are met in many situations, but certainly not all situations. Today we will look at a scenario that is quite different, and that needs a very different approach - now called Tomasi-Kanade factorization[1]. Here is what we assume:

- F image frames (not just 2). Normally one collects many frames using a video camera.
- N corresponding points (x_{ki}, y_{ki}), $i = 1, /dots, N$ that are visible in each frame k. Typically, these points are detected and tracked from frame to frame. We have not discussed the problem of *tracking* a point – but we have discussed how to find interest points (Harris corners) and how to register two images. So you can imagine that we could find the motion of an interest point from one frame to the next. That is basically the tracking problem.

  The N points have *fixed* 3D coordinates $\{(X_i, Y_i, Z_i), i = 1, ..n\}$ which are expressed in unknown object coordinate system. Without loss of generality, we assume

  $$\sum_{i=1}^{N} \begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{1}$$

  that is, we assume the origin is the centroid of the object. (We need to divide by N to get this, but the right side is zero so dividing by N doesn't do anything to it.)
- In each of the F frames, the average depth of the points is about Z_o and the range of depths is small, relative to Z_o. An example is that you are holding a small object at arm's length, you rotate it in your hand. Another example is shown below. (This was photographed using f = 80 mm, which gives a small field of view.)



[1]Tomasi and Kanade, "Shape and motion from image streams: a factorization method", IJCV 1992

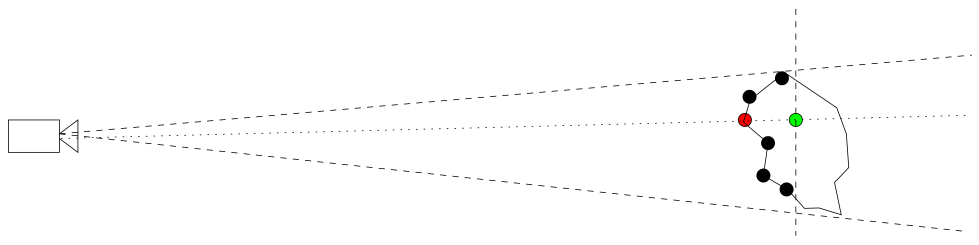
- The camera's $\mathbf{K}$ matrix is known. So, we can back-project points from pixel coordinates (x_{ki}, y_{ki}) to points on frame k's $Z = Z_o$ plane, and *define*:

$$\begin{bmatrix} X_{ki} \\ Y_{ki} \\ Z_0 \end{bmatrix} \equiv \frac{Z_o}{f}\mathbf{K}^{-1} \begin{bmatrix} x_{ki} \\ y_{ki} \\ f \end{bmatrix} \qquad (2)$$

The $\mathbf{K}^{-1}$ mapping takes pixel coordinates to projection plane coordinates $(Z = f)$ and then multiplying by $\frac{Z_o}{f}$ rescales so all points are at depth Z_o in camera coordinates.

NOTE: the true point positions are not at constant depth, so there is an approximation being made here: the X_{ki} and Y_{ki} values are only approximately equal to the true X, Y positions of the points (represented in camera k's coordinates), and of course Z_0 is only an approximation to the true depths of point i in camera k's coordinates.

This approximation is illustrated in the sketch below. We are assuming that the green point's coordinates are approximately the same as the red point's coordinates. In particular, below we will only use the X_{ki} and Y_{ki} values. This approximation of the X_{ki} and Y_{ki} values gets better if Z_0 is large compared to the range of depths within the object, since the projection directions (to the camera center) become near parallel with the optical axis. Recall that this is precisely what was assumed on page 1.



Problem

We have set up the scenario. Now let's define the problem we want to solve. Let $\mathbf{A}$ be the image measurement matrix, which is constructed from the 2×1 matrices $\begin{bmatrix} X_{ki} \\ Y_{ki} \end{bmatrix}$, i.e. point i seen in the k^{th} frame, $k = 1, \ldots, m$. $\mathbf{A}$ is $2F \times N$ data matrix. Think of it as two $F \times N$ matrices, one for the X_{ki} values and one for the Y_{ki} values.

Each frame k defines its own mapping $\mathbf{R}_k[\ \mathbf{I} \mid -\mathbf{C}_k\ ]$ from the object XYZ coordinate system into the k^{th} camera coordinate system. So, for object point i and frame k, we have

$$\begin{bmatrix} X_{ki} \\ Y_{ki} \\ Z_0 \end{bmatrix} \approx \mathbf{R}_k[\ \mathbf{I} \mid -\mathbf{C}_k\ ] \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}$$

The three rows of $\mathbf{R}_k$ are the camera's unit X, Y, Z axes, expressed in the object's coordinate frame – in particular, the third row of $\mathbf{R}_k$ is the camera's Z axis, and the third element of $\mathbf{RC}_k$ is the depth of the origin of the object's coordinate system $(X, Y, Z) = (0, 0, 0)$ and this is our Z_0. The model is approximating the Z values of all points as having this depth. (See Eq. (2)).

The approximation for the first two rows is very good if the assumptions on page 1 are met, whereas the approximation in the third row is not good, since it ignores the depth variations of the points in camera k. Because the third row's approximation is not good, we will ignore it.

So finally, the problem to be solved is this: Given the $2F \times N$ matrix $\mathbf{A}$, decompose it into the variables shown on the right side, namely the "motion" $\mathbf{R}_k$ (which is just the orientations of the camera at frame k) and the scene "structure", i.e. the 3D positions (X_i, Y_i, Z_i) of the N points, as expressed in the object coordinate frame.

Solution (Factorization)

Let $\tilde{\mathbf{R}}_k$ be the 2×3 matrix which is the first two rows of $\mathbf{R}_k$. For any camera k,

$$\sum_{i=1,..N} \begin{bmatrix} X_{ki} \\ Y_{ki} \end{bmatrix} = \sum_{i=1,..N} \tilde{\mathbf{R}}_k [\ \mathbf{I} \mid -\mathbf{C}_k\ ] \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}_{4\times N} = \tilde{\mathbf{R}}_k [\ \mathbf{I} \mid -\mathbf{C}_k\ ] \sum_{i=1,..N} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}_{4\times N}$$

Applying Eq. (1) to the right side, we get

$$\sum_{i=1,..N} \begin{bmatrix} X_{ki} \\ Y_{ki} \end{bmatrix}_{2m\times K} = -N \tilde{\mathbf{R}}_{\mathbf{k}} \mathbf{C}_{\mathbf{k}\ 2\times 1}$$

Thus, for each camera k, we can compute $\tilde{\mathbf{R}}_{\mathbf{k}}\mathbf{C}_{\mathbf{k}}$, namely by taking the average of the (X_{ki}, Y_{ki}) values over all i:

$$\begin{bmatrix} \bar{X}_k \\ \bar{Y}_k \end{bmatrix}_{2m\times K} \equiv \frac{1}{N} \sum_{i=1,..N} \begin{bmatrix} X_{ki} \\ Y_{ki} \end{bmatrix}_{2m\times K} = -\tilde{\mathbf{R}}_{\mathbf{k}} \mathbf{C}_{\mathbf{k}\ 2\times 1}$$

This gives:

$$\begin{bmatrix} X_{ki} - \bar{X}_k \\ Y_{ki} - \bar{Y}_k \end{bmatrix}_{2\times N} = [\tilde{\mathbf{R}}_k]_{2\times 3} \begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix}_{3\times N} \tag{3}$$

We have two equations for each frame k, so we stack them to get a matrix product $2F \times N = (2F \times 3) \times (3 \times N)$. That is, we have a $2F \times N$ data matrix (left side) which is (approximately!) factored into the product of a matrix whose rows are the true X, Y camera axes of the k cameras, and a matrix whose columns are the true 3D point points written in object coordinates.

If there were no image noise and if the approximation in the projection model were exact, then the data matrix on the left would have rank 3, that is, the column space would span a (at most) 3 dimensional space only. To see this, note that the right side of Eq. (3) maps from $\Re^N$ to $\Re^3$ and then from $\Re^3$ to $\Re^{2m}$. The column space of the matrix of $\mathbf{R}$ vectors has dimension at most 3 and so this limits the dimension of the vectors reached by the mapping (i..e. the range).

We would like to estimate these $\tilde{\mathbf{R}}$ matrices as well as the points (X_i, Y_i, Z_i). How can we do it?

Let the $2F \times N$ data matrix on the left side of Eq. (3) be $\mathbf{A}$. We can write it using the SVD

$$\mathbf{A} = \mathbf{U}_{2F\times N} \mathbf{\Lambda}_{N\times N} \mathbf{V}^T_{N\times N}.$$

If the model were perfect, then **A** would have rank 3, i.e. only three of the singular values would non-zero. We look for a rank 3 approximation (the factorization) so we first verify that the 4th largest singular value is much smaller than the third largest and, if this is the case, we keep only use the largest 3 singular values and the corresponding eigenvectors:

$$\mathbf{A} \approx \mathbf{U}^*_{2F\times 3}\mathbf{\Lambda}^*_{3\times 3}\mathbf{V}^{*T}_{3\times N}$$

As long as the ignored singular values are small, the column space of **A** will be approximately the same as the column space of $\mathbf{U}^*$.

Notice that the $\mathbf{U}^*$ matrix and the $(\mathbf{\Lambda}^*\mathbf{V}^{*T})$ have the same dimensions as the rotation (motion) and (X, Y, Z) (structure) matrices earlier. However, we cannot just assign these as our solution since there is no reason why rows $2k-1$ and $2k$ of $\mathbf{U}^*$ should be orthonormal. And we *need* the two rows of $\tilde{\mathbf{R}}_k$ to be orthonormal.

But notice that

$$\mathbf{U}^*\mathbf{\Lambda}^*\mathbf{V}^{*T} = \mathbf{U}^*\mathbf{Q}\mathbf{Q}^{-1}\mathbf{\Lambda}^*\mathbf{V}^{*T}$$

for any invertible matrix 3×3 **Q**. Such a **Q** matrix takes three linear combinations of the columns of $\mathbf{U}^*$, namely each column of **Q** defines a linear combination of the columns of $\mathbf{U}^*$. We would like to take linear combinations of the three columns of $\mathbf{U}^*$ such that the F pairs of rows of $\mathbf{U}^*\mathbf{Q}$ *are* orthonormal, or as close to orthonormal as we can make them.

Specifically, let $\mathbf{u}_{1i}$ and $\mathbf{u}_{2i}$ be the i^{th} pair of 1×3 rows of $\mathbf{U}^*$. We want to choose **Q** such that

$$(\mathbf{u}_{1i}\mathbf{Q})\cdot(\mathbf{u}_{1i}\mathbf{Q}) = 1$$

$$(\mathbf{u}_{2i}\mathbf{Q})\cdot(\mathbf{u}_{2i}\mathbf{Q}) = 1$$

$$(\mathbf{u}_{1i}\mathbf{Q})\cdot\mathbf{u}_{2i}\mathbf{Q} = 0$$

But notice this defines $3F$ equations that are second order in the 9 elements of **Q**. We want to find a **Q** that simultaneously solves these three equations – or at least solves them as close as possible. (We have more equations than unknowns and we have made approximations along the way, so we don't expect an exact solution.) This requires we solve a *non-linear* least squares problem. (See next page.)

One final point (not mentioned in class). Even if we use the factorization on the right side of

$$\mathbf{U}^*\mathbf{\Lambda}^*\mathbf{V}^{*T} = (\mathbf{U}^*\mathbf{Q})(\mathbf{Q}^{-1}\mathbf{\Lambda}^*\mathbf{V}^{*T})$$

there is still an ambiguity since we can always insert any 3×3 orthonormal matrix **W**

$$(\mathbf{U}^*\mathbf{Q})(\mathbf{Q}^{-1}\mathbf{\Lambda}^*\mathbf{V}^{*T})(\mathbf{U}^*\mathbf{Q}\mathbf{W})(\mathbf{W}^T\mathbf{Q}^{-1}\mathbf{\Lambda}^*\mathbf{V}^{*T})$$

and this will be an equally good solution/factorization i.e. since the quadratic constraints above are unaffected by the **W**. What are these extra rotation/reflection degrees of freedom? They just allow for the fact that we cannot solve for the orientation of the object coordinate XYZ axes – they are always known only up to a rotation.

Non-linear least squares (using Newton's method)

We write the $3F$ equations above as:

$$\vec{f}(\mathbf{Q}) = \mathbf{0}$$

Note: we have brought the 1's on the right side to the left side. We would like to find a 3×3 matrix $\mathbf{Q}$ that satisfies them. In our example, the equations are each quadratic in the 9 variables of $\mathbf{Q}$ and the coefficients depend on the $\mathbf{U}^*$ values that were obtained from the SVD.

We make an initial guess for $\mathbf{Q}$, namely we set it to the identity matrix, $\mathbf{Q}_0 = \mathbf{I}$. If $\vec{f}(\mathbf{Q}_0) = \mathbf{0}$ then we are done. (This won't happen.) So, we ask how to change the matrix $\mathbf{Q}_0$ by adding some Δ, and updating our estimate to $\mathbf{Q}_0 + \Delta$. How do we choose Δ such that the solution is better ?

We want $\vec{f}(\mathbf{Q}_0 + \Delta) = \mathbf{0}$ so we try to find Δ by approximating $\vec{f}()$ as linear in a neighborhood of $\mathbf{Q}_0$. We compute the $3F \times 9$ Jacobian matrix, namely the matrix of partical derivatives, and we evaluate it at $\mathbf{Q}_0$,

$$\mathbf{J} \equiv \frac{\partial \vec{f}}{\partial \mathbf{Q}} \,|_{\mathbf{Q}_0}.$$

The linear approximation is

$$\vec{f}(\mathbf{Q}_0 + \Delta) \approx \vec{f}(\mathbf{Q}_0) + \mathbf{J}\Delta$$

We then require $\vec{f}(\mathbf{Q}_0 + \Delta) = 0$, and seek at Δ such that

$$-\vec{f}(\mathbf{Q}_0) \approx \mathbf{J}\Delta$$

This is now a least squares problem, since we can evaluate the $\vec{f}(\mathbf{Q}_0)$ and the Jacobian $\mathbf{J}|_{\mathbf{Q}_0}$ so they are just a vector and matrix of numbers, respectively.

The Δ that we get will not solve $\vec{f}(\mathbf{Q}_0 + \Delta) = 0$ exactly of course, for two reasons: first, $\vec{f}()$ is quadratic, not linear; second, least squares gives a solution that minimizes the sum of squared errors, but this minimum might not be zero.

So, since we don't have an exact solution, we iterate:

$$\mathbf{Q}_{j+1} = \mathbf{Q}_j + \Delta_j$$

and keep going until $\mathbf{Q}_{j+1} \approx \mathbf{Q}_j$. If we get $\vec{f}(\mathbf{Q}_{j+1}) \approx \vec{0}$, then we have a good solution.

[ASIDE: Does this algorithm alwyas converge, and does it converge to a good solution? It would seem so, since quadratic functions are usually well-behaved. However, I cannot offer any guarentees. The published paper by Tomasi and Kanade doesn't say anything about it, nor do textbooks that summarize the paper. (Perhaps it is in Tomasi's PhD thesis?) I spoke to Prof. Xiaowen Chang about this problem – but at first glance it was not obvious to him either that it was always well behaved.]