**EECS C106B / 206B Robotic Manipulation and Interaction** **Spring 2020**

# Lecture 15: Geometry and Algebra of Multiple Views

*Scribes: Derek Liu, Sunay Poole*

## 15.1 Summary of reconstruction transforms up to now

| | Calibrated case | Uncalibrated case |
|---|---|---|
| Image point | $x$ | $x' = Kx$ |
| Camera (motion) | $g = (R, T)$ | $g' = (KRK^{-1}, KT)$ |
| Epipolar constraint | $x_2^T E x_1 = 0$ | $(x_2')^T F x_1' = 0$ |
| Fundamental matrix | $E = \widehat{T}r$ | $F = \widehat{T}'KRK^{-1}, T' = KT$ |
| Epipoles | $Ee_1 = 0, e_2^T E = 0$ | $Fe_1 = 0, e_2^T F = 0$ |
| Epipolar lines | $l_1 = E^T x_2, l_2 = Ex_1$ | $l_1 = F^T x_2', l_2 = Fx_1'$ |
| Decomposition | $E \mapsto [R, T]$ | $F \mapsto [\widehat{(T')^T}F, T']$ |
| Reconstruction | Euclidean: $X_e$ | Projective: $X_p = HX_e$ |

This is a continuation from the last lecture, which talks about reconstructions from both calibrated and uncalibrated cameras and how to work between the two.

Calibration is a very important topic - in robotics, generally one of the most important things is to calibrate your system well. Although last lecture we started by talking about the calibrated case, often times you are presented with an uncalibrated problem where you don't know $K$, and thus can really only count on the epipolar geometry and constraint $(x_2')^T F x_1' = 0$, where instead of the essential matrix $E$ you are working with a transformed version $F$. The geometric picture is still the same, as corresponding features from two views still must be coplanar per the epipolar geometry. Now, in that setting, before you can recover any information about calibration, the best you could do is to reconstruct a structure of the scene that is consistent with the uncalibrated views, using the equations given in the above table.

The important fact is that yes, you can reconstruct an $X_p$, but it will be different from the true structure $X_e$ by not a rigid body transformation per se, but rather a more general projective transformation $H$ such that $X_p = HX_e$.

| | Camera projection | 3-D structure |
|---|---|---|
| Euclidean | $\Lambda_{1e} = [K, 0], \Lambda_{2e} = [KR, KT]$ | $X_e = g_e X = \begin{bmatrix} R_e & T_e \\ 0 & 1 \end{bmatrix} X$ |
| Affine | $\Lambda_{2a} = [KRK^{-1}, KT],$ | $X_a = H_a X_e = \begin{bmatrix} K & 0 \\ 0 & 1 \end{bmatrix} X_e$ |
| Projective | $\Lambda_{2p} = [KRK^{-1} + KTv^T, v_4 KT],$ | $X_p = H_p X_a = \begin{bmatrix} I & 0 \\ -v^T v_4^{-1} & v_4^{-1} \end{bmatrix} X_a$ |

For some applications $X_p$ is okay. However, if you want to work in Euclidean terms, you want $X_e$. We know that it differs from $X_e$ by $H$, which can be described in two specific geometric steps. First, we notice that some of the lines in the projective reconstruction that should be parallel are not. Two parallel lines are supposed to intersect at infinity – yet here, these lines intersect at finite points.

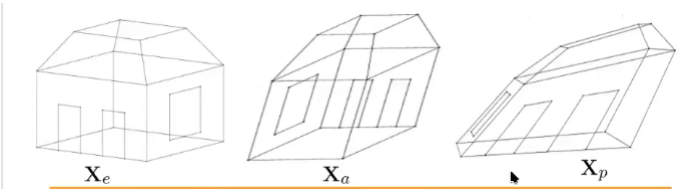


Figure 15.1: A visualisation of different reconstructions. Affine distortion produces something like $X_a$, while projective distortion produces something like $X_p$

Our conversion back to Euclidean reconstruction, $H^{-1}$, can be expressed as

$$H^{-1} = \begin{bmatrix} K_1^{-1} & 0 \\ v^T & v_4 \end{bmatrix}$$

which can then be split up into an affine and projective component $H_a^{-1}$ and $H_p^{-1}$ respectively:

$$H^{-1} = \begin{bmatrix} K_1^{-1} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} I & 0 \\ v^T & v_4 \end{bmatrix}$$

The projective component $H_p$, specifically, is what is mapping lines that should intersect at infinity to lines that intersect at some finite point.

But how do we compute:

$$H_p^{-1} = \begin{bmatrix} I & 0 \\ v^T & v_4 \end{bmatrix}$$

We know that it maps the points $[v, v_4]^T X_p = 0$ to points with affine coordinates $X_a = [X, Y, Z, 0]^T$.

Using three vanishing points (so 3 different $X_p$), one can solve for $[v, v_4] = [v_1, v_2, v_3, v_4]$ (assuming a fixed scale), and use that to set up $H_p^-1$ go to an affine structure, where one would attempt to recover the orthogonality of the original structure.

## 15.2 Calibration

You can use the fact that both 3-D and 2-D coordinates of feature points on a pre-fabricated object (e.g., a cube) are known. This cube can be used for both finding the $K$ matrix but also correcting other distortions. Algorithms to do so using things like checkerboards exist using optimization.

Figure 15.2: Example of such a calibration cube

In general, the aim of calibration is to make "straight lines be straight, parallel lines be parallel, and orthogonal angles be orthogonal". In general this can be quite application specific – many of the specifics will not be covered in 106B or related course material, and one would have to apply the tools given in novel ways to formulate a solution. But in general, trying to apply these principles is the way to go.

Even if you pre-calibrate your camera, you still have to worry about the focus/focal length of the camera, or other miscellaneous error.

This is where structural information, such as vanishing points can come in handy, as you are given a bunch of these in nature.

## 15.3 Vanishing points, the intersection of supposedly parallel lines

$$v_i = l_1 \times l_2 = \widehat{l_1} l_2$$

If you take vanishing points of three orthogonal directions $v_i = KRe_i$ for $i = 1, 2, 3$, we can see that

$$v_i^T S v_j = v_i^T K^{-T} K^{-1} v_j = e_i^T R^T R e_j = e_i^T e_j = 0$$

for $i \neq j$, and some matrix $S = K^{-T}K$ constrained by the three vanishing points. Finding $S$ allows you to derive $K$. These vanishing points are not hard to find in say, robotics applications.

Sometimes, you are given more information about the situation, for example, take the case where the cameras are all only related to each other by a pure rotation (like on a pivot), so you'd get

$$\lambda_2 K x_2 = \lambda_1 K R K^{-1} K x_1$$

and

$$\widehat{x_2'} K R K{-}1 x_1' = 0$$

The conjugate rotation $C = KRK^{-1}$ could be estimated from external dataand one could also have the three linear constraints $S^{-1} - CS^{-1}C^T = 0$ where $S^{-1} = KK^T$

Unfortunately, in practice, despite the mathematical niceness of the situation, it's basically impossible to use something like this well in practice because it's hard to put a camera on a perfect pure rotation. But, you can also get more constraints from the fundamental matrix $F = K^{-T}\hat{T}RK^{-1} = \hat{T}'KRK^{-1}$ as it must satisfy the Kruppa's equations

$$FKK^TF^T = \hat{T}'KK^T\hat{T}'^T$$

if the fundamental matrix is known up to scale

$$FKK^TF^T = \lambda^2\hat{T}'KK^T\hat{T}'^T$$

and this yields two nonlinear constraints on $S^{-1} = KK^T$. (These constraints can be linear if you know more about how the cameras are moving.)

Although for robotics applications, it's more important to get the pre-calibration right and figure out the few loose parameters later.

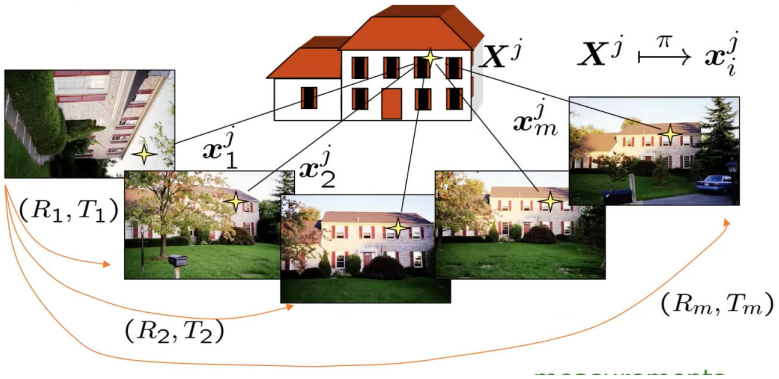


Figure 15.3: An illustration of multiple points in multiple camera images mapping to one point on a house.

## 15.4 Multiple-view geometry

Two-view geometry allows us to solve for the relative pose between two cameras and also triangulate points.

Multiple views will allow for more robust structure estimation (such as depth), but won't necessarily improve measurements for motion, as more views adds more unknowns.

There are also technical reasons to pursue more than two views. Uncalibrated camera systems would need a third view. Analysis of line features and consistent scaling also require a third view for that last constraint.

Multiple views can also do feature tracking at high framerates, and estimate motion at low framerates.

To put this more into context, take a problem where you are given corresponding images of "features" of, say, a house, in multiple images. We wish to recover the camera motion, camera calibration, and object structure.

Say in our image system, the $j$-th feature in the $i$-th camera is $x_i^j$, and each camera has an associated rotation and translation $R_i$ and $T_i$, and there exists some function $\pi$ that maps a point on the actual house $X_j$ to various $x_i^j$. We would wish to minimize the optimization problem:

$$F(R_i, T_i, X^j) = \sum_{j=1}^{n} \sum_{i=1}^{m} \|x_i^j - \pi(R_i X^j + T_i)\|^2$$

## 15.5 The various ways of formulating multi-view constraints

### 15.5.1 General intuition

As we will see, there are many different ways to formulate the geometry of the multi-view problem in algebra, but between them all, they all express the same kind of constraint.

If one were to cast rays from the image source onto say, the ends of a line feature in an image, and consider the resulting plane (the "pre-image"), these planes from multiple cameras should all intersect at the physical feature that say, corresponds to a line as viewed from a camera. This is a core constraint that many mathematical formulations around image processing revolve around.

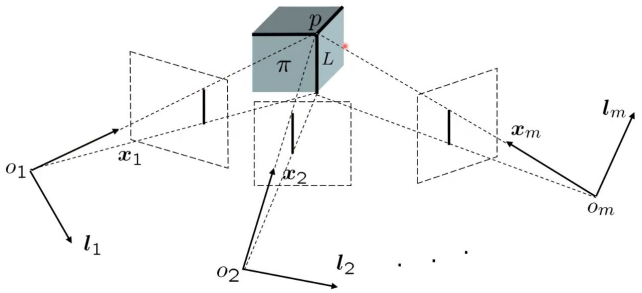


Figure 15.4: The planes formed by the "pre-images" (shown here as triangular rays) all intersect at the actual edge of the cube that corresponds to the lines in each image. This is the type of constraint that is expressed in the algebra.

### 15.5.2 A formulation from the textbook

When we consider a point in 3-D with its homogenous coordinates $X = [X, Y, Z, W]^T \in \mathbb{R}^4$, and corresponding homogeneous coordinates of its 2-D camera image $x = [x, y, z]^T \in \mathbb{R}^3$ where $W$ and $z$ are usually 1, we can consider its projection of $X$ to the image plane as $\lambda(t)x(t) = \Pi(t)X$, where $\lambda(t) \in \mathbb{R}$ and $\Pi(t) = [R(t), T(t)] \in \mathbb{R}^{3\times 4}$. (We assume $\Pi(t)$ deals with rotation and translation as applied from a calibrated camera.)

Then, without loss of generality, we choose the frame of say the first camera view as a reference, such that

$$\lambda_i x_i = \lambda_1 R_i x_1 + T_i$$

Applying the cross product of some point $x_i$ to both sides and rearranging into a dot product yields the equivalent expression

$$\begin{bmatrix} A\widehat{x_i}R_i x_1 & \widehat{x_i}T_i \end{bmatrix} \begin{bmatrix} \lambda_1 \\ 1 \end{bmatrix} = 0$$

Then, for all other frames of reference, we could generalize this and create a multiple view matrix

$$M_p = \begin{bmatrix} \widehat{x_2}R_2x_1 & \widehat{x_2}T_2 \\ \widehat{x_3}R_3x_1 & \widehat{x_3}T_3 \\ \vdots & \vdots \\ \widehat{x_m}R_mx_1 & \widehat{x_m}T_m \end{bmatrix} \in \mathbb{R}^{3(m-1)\times 2}$$

where $\text{rank}(M_p) \leq 1$ – it is 1 in the generic case (as all rows when multiplied with $\begin{bmatrix} \lambda & 1 \end{bmatrix}^T$ will produce 0 and thus be linearly dependent) and 0 in the degenerate case. Interestingly, this only depends on the single point $x_1$ and the camera orientations $R_i$ and $T_i$.

In the rank 1 case, the $m$ rays from each camera's origin through each of the respective $x_i$s must intersect at a single point.

The rank 1 case also has some interesting properties as well, for example, the implied linear dependence between $\widehat{x_i}R_ix_1$ and $\widehat{x_i}T_i$ is equivalent to the epipolar constraint $x_i^T\widehat{T_i}R_ix_1 = 0$

Additionally, if we were to take say two rows in the matrix for points $i$ and $j$, say

$$M_{p,(i,j)} = \begin{bmatrix} \widehat{x_i} R_i x_1 & \widehat{x_i} T_i \\ \widehat{x_j} R_j x_1 & \widehat{x_j} T_j \end{bmatrix}$$

We'd also note that $M_p$ being rank 1 would also have $\det(M_{p,(i,j)}) = 0$, which is equivalent to the trilinear constraint $\widehat{x_i}(R_i x_1 T_j^T - T_i x_1^T R_j^T)\widehat{x_j} = 0$. This can be extended to four or more points with quadriliinear constraints and so on. As the saying goes, geometry is drawn algebra, and algebra is drawn geometry.

Other ways to formulate the constraints of m corresponding images of n points $x_i^J$

$$\lambda_i^j x_i^j = \Lambda_i X^j$$

where $\Lambda_i = [R_i, T_i]$

include as this form presented by Heyden et. al.

$$H_p = \begin{bmatrix} \Lambda_1 & x_1 & 0 & \dots & 0 \\ \Lambda_2 & 0 & x_2 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ \Lambda_1 & x_1 & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} X \\ -\lambda_1 \\ \vdots \\ -\lambda_m \end{bmatrix} = 0, H_p \in \mathbb{R}^{3m \times (m+4)}, \det\left(H_{(m+4)\times(m+4)}\right) = 0$$

And Faugeras et. al.

$$F_p = \begin{bmatrix} \widehat{x_1}\Lambda_1 \\ \widehat{x_2}\Lambda_2 \\ \vdots \\ \widehat{x_m}\Lambda_m \end{bmatrix} X = 0, F_p \in \mathbb{R}^{3m \times 4}, \det(F_{4\times 4}) = 0$$

There are tradeoffs to each formulation, but in the end they are algebraically equivalent. The one presented earlier is the one used in the textbook.

## 15.6 Reconstruction algorithm for point features

**Given $m$ images of $n$ points** $(x_1^J, \dots, x_i^j, \dots, x_m^j)_{j=1,\dots,n}^{i=1,\dots,m}$

We get linear equations relating points given rotation/translations of cameras:

$$\lambda^j \begin{bmatrix} \widehat{x_2^j} R_2 x_1^j \\ \widehat{x_3^j} R_3 x_1^j \\ \vdots \\ \widehat{x_m^j} R_m x_1^j \end{bmatrix} + \begin{bmatrix} \widehat{x}_2^j T_2 \\ \widehat{x}_3^j T_3 \\ \vdots \\ \widehat{x}_m^j T_m \end{bmatrix} = 0, \in \mathbb{R}^{3(m-1)\times 1}$$

and linear equations relating rotations/translations given points:

$$P_i \begin{bmatrix} R_i^s \\ T_i^s \end{bmatrix} = \begin{bmatrix} \lambda^1 x_1^1 \otimes \widehat{x_i^1} & \widehat{x_i^1} \\ \lambda^2 x_1^2 \otimes \widehat{x_i^2} & \widehat{x_i^2} \\ \vdots & \vdots \\ \lambda^n x_1^n \otimes \widehat{x_i^n} & \widehat{x_i^n} \end{bmatrix} \begin{bmatrix} R_i^s \\ T_i^s \end{bmatrix} = 0, \in \mathbb{R}^{3n \times 1}$$

where $\otimes$ is the Kronecker (element-wise) product.

If $n \geq 6$, then in general $P_i$ will have rank 11.

If we're given $m$ images of $n > 6$ points, for the $j$-th point, we can setup the relation

$$\begin{bmatrix} \widehat{x_2^j} R_2 x_1^j & \widehat{x_2^j} T_2 \\ \widehat{x_3^j} R_3 x_1^j & \widehat{x_3^j} T_3 \\ \widehat{x_m^j} R_m x_1^j & \widehat{x_m^j} T_m \end{bmatrix} \begin{bmatrix} \lambda^j \\ 1 \end{bmatrix} = 0$$

and using the SVD, extract a $\lambda^{j^s}$

Similarly, for the $i$-th image, applying the SVD/Moore-Penrose pseudoinverse to:

$$\begin{bmatrix} \lambda^1 x_1^1 \otimes \widehat{x_i^1} & \widehat{x_i^1} \\ \lambda^2 x_1^2 \otimes \widehat{x_i^2} & \widehat{x_i^2} \\ \vdots & \vdots \\ \lambda^n x_1^n \otimes \widehat{x_i^n} & \widehat{x_i^n} \end{bmatrix} \begin{bmatrix} R_i^s \\ T_i^s \end{bmatrix} = 0$$

can also extract a pose $(R_i^s, T_i^s$. Now, of course, these may not be accurate on the first try, especially if some of the camera rotations/translations or the $\lambda$s are wrong. By alternating between these two problems and using the result of one to adjust the other, this sort of bilinear optimization can be used conceptually to converge to a result. (The proof of convergence is outside the lecture's scope.)

So, we can write this algorithm down in a series of steps:

1. Initialization
   - Set k = 0
   - Compute $(R_2, T_2)$ using the 8-point algorithm
   - Compute $\lambda^j = \lambda_k^j$ and normalize so that $\lambda_k^1 = 1$
2. Compute $(\tilde{R}_i, \tilde{T}_i)$ as the nullspace of $P_{i=2,\ldots,m}$
3. Compute new $\lambda^j = \lambda_{k+1}^j$ as the nullspace of $M^j, j = 1, \ldots, n$, and once again normalize such that $\lambda_{k+1}^1 = 1$
4. If $\|\lambda_k - \lambda_{k+1}\| \leq \epsilon$ terminate, else $k = k + 1$ and goto step 2.

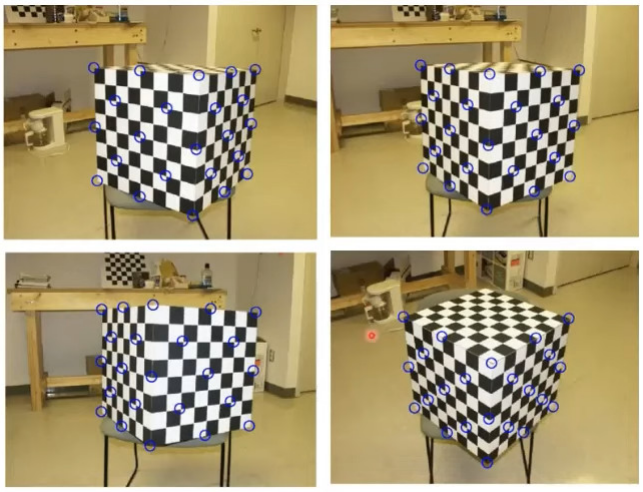

Figure 15.5: As it turns out, this works out pretty well in practice!

## Appendix

The Overleaf for these notes is at `https://www.overleaf.com/read/yrtkvrrmpnnt`, if you wish to take these notes and improve them.