Talia Weiss – on camera calibrations (2022-01-13)

The pinhole camera model is as follows:

A point on an image is noted by its pixel coordinates, (u, v)

This image point is a projection of a real point in the world, given by (X, Y, Z).

Lets start with one camera. The pixel coordinate (u, v) in the coordinate system relative to the camera is given as (X_c, Y_c, Z_c).

In general, we assume that there is a mapping between (u, v) and(X_c, Y_c, Z_c) given by:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \boldsymbol{K}_{3x3} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \text{ or } s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \boldsymbol{K}_{3x3} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}$$

Here, the w or s represents the fact that with a single camera, without knowing the distance from the camera to the object, we can only calculate this correspondence with respect to an arbitrary scale

$\boldsymbol{K}_{3x3} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$ is known as the intrinsic matrix and contains the focal lengths f_x, f_y (these are equal for a camera with square pixels, usually), and the camera's principle point (c_x, c_y) which is usually the center of the image plane – if a camera is $N\ x\ M$ pixels, $c_x \approx \frac{N}{2}, c_y \approx \frac{M}{2}$

However in general, we want to map the pixel coordinates (u, v) to 3D coordinates in an arbitrary coordinate system, not just the camera's. (This is required for DLT, for example). In that case, we can map the 3D coordinate in the camera reference system to some world reference system:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R_{3x3}} & \boldsymbol{T_{(3x1)}} \\ \boldsymbol{0} & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{12} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

Where $\boldsymbol{R}$ is the rotations between the world coordinate system and the cameras, and $\boldsymbol{T}$ is the translation between the world coordinate systems and the camera.

This leads us to the final pinhole camera model:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \boldsymbol{K}\ [\boldsymbol{R} \mid \boldsymbol{T}] \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

Note that in general, $\boldsymbol{K}\ [\boldsymbol{R} \mid \boldsymbol{T}]$ can simply per represented as the perspective matrix $\boldsymbol{P_{3x4}}$:

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \boldsymbol{P}_{3x4}\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix}\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

The perspective matrix may have 12 components, but only 11 are independent (because math) (it's a rank 3 matrix)

Notes on DLT:

DLT, or direct linear transform, is a way to solve for either (1) a 2D – 2D point correspondence or (2) a 3D to 2D point correspondence.

DLTdv8 from Ty Hedrick only takes DLT parameters as an input for calibrated cameras, as opposed to intrinsic and extrinsic matrices. What do you do when you only have what OpenCV spits out, which is for two cameras, their intrinsic matrices $\boldsymbol{K_1}, \boldsymbol{K_2}$ and the extrinsic correspondence between camera 1 and camera 2, [$\boldsymbol{R}$ | $\boldsymbol{T}$]

The DLT coefficients (ignoring distortion for now) is defined as the mapping from world coordinates to image pixel locations for a single camera. Each camera has its own set of coefficients, of which there are at least 11. The DLT coefficients, $L_1 \rightarrow L_{11}$ are used to calculate the pixel coordinates from a 3D location as follows:

$$u = \frac{L_1 X_w + L_2 Y_w + L_3 Z_w + L_4}{L_9 X_w + L_{10} Y_w + L_{11} Z_w + 1}$$

$$v = \frac{L_5 X_w + L_6 Y_w + L_7 Z_w + L_8}{L_9 X_w + L_{10} Y_w + L_{11} Z_w + 1}$$

The first thing you might want to use DLTdv8 to do is draw epipolar lines for you, so you can tell how good your calibration is while point clicking, and maybe help point click partially obscured areas. Now, algebraically to solve for the epipolar line one only needs the fundamental matrix $\boldsymbol{F_{3x3}}$ (which is rank 2), which OpenCV also gives us.

However, DLTdv8 directly computes a slope and intercept for the epipolar line using $L_1 \rightarrow L_{11}$ (of camera 1) by making up two Z_w coordinate for a point (u, v) on image(camera) 1, calculating the (X_w, Y_w) for that pixel coordinate using the equations above, to get two possible 3D coordinates for the $(u, v)_{c1}$. Then, using the same equations as above, but this time using the (X_w, Y_w, Z_w) and $L_1 \rightarrow L_{11}$ for camera 2 to calculate 2 corresponding pixel coordinates on image plane 2 $(u, v)_{c2}$. Using these two pixel coordinates for camera 2, the slope and intercept are then calculated for drawing an epipolar line)

This is implemented in partialdlt.m

So, how do you get DLT coordinates just from the intrinsic and extrinsic matrix? Well remember the perspective matrix?

$$s\begin{bmatrix}u\\v\\1\end{bmatrix} = \boldsymbol{P}_{3x4}\begin{bmatrix}X_w\\Y_w\\Z_w\\1\end{bmatrix} = \begin{bmatrix}p_{11} & p_{12} & p_{13} & p_{14}\\p_{21} & p_{22} & p_{23} & p_{24}\\p_{31} & p_{32} & p_{33} & p_{34}\end{bmatrix}\begin{bmatrix}X_w\\Y_w\\Z_w\\1\end{bmatrix}$$

$$s\begin{bmatrix}u\\v\\1\end{bmatrix} = \begin{bmatrix}p_{11}X_w + p_{12}Y_w + p_{13}Z_w + p_{14}\\p_{21}X_w + p_{22}Y_w + p_{23}Z_w + p_{24}\\p_{31}X_w + p_{32}Y_w + p_{33}Z_w + p_{34}\end{bmatrix}$$

Now…doesn't that look familiar?

As this is just to calculate epipolar lines, the scale s here doesn't matter. If we set $s' = s/p_{34}$ we can get:

$$s'\begin{bmatrix}u\\v\\1\end{bmatrix} = \begin{bmatrix}\frac{p_{11}}{p_{34}}X_w + \frac{p_{12}}{p_{34}}Y_w + \frac{p_{13}}{p_{34}}Z_w + \frac{p_{14}}{p_{34}}\\\frac{p_{21}}{p_{34}}X_w + \frac{p_{22}}{p_{34}}Y_w + \frac{p_{23}}{p_{34}}Z_w + \frac{p_{24}}{p_{34}}\\\frac{p_{31}}{p_{34}}X_w + \frac{p_{32}}{p_{34}}Y_w + \frac{p_{33}}{p_{34}}Z_w + 1\end{bmatrix}$$

We now how a direct relationship between $\boldsymbol{P}$ and $L_1 \rightarrow L_{11}$

And remember, $\boldsymbol{P} = \boldsymbol{K}\,[\,\boldsymbol{R} \mid \boldsymbol{T}\,]$

We just have one problem left: we still don't have a world coordinate system. OpenCV will spit out an extrinsic matrix for camera 1 in relation to camera 2. So actually all we have are: $K_1, K_2, R_{2\rightarrow1}, T_{2\rightarrow1}$

However, if you just say $\begin{bmatrix}u\\v\\w\end{bmatrix} = K\begin{bmatrix}X_c\\Y_c\\Z_c\end{bmatrix}$, The DLT coefficients will not work for getting you epipolar lines.
Instead, make up a world coordinate system:

$$\begin{bmatrix}X_c\\Y_c\\Z_c\end{bmatrix} = [\,\boldsymbol{R}_{w\rightarrow c1} \mid \boldsymbol{T}_{\boldsymbol{w}\rightarrow\boldsymbol{c1}}]\begin{bmatrix}X_w\\Y_w\\Z_w\end{bmatrix}$$

$$\begin{bmatrix}u\\v\\w\end{bmatrix}_{c1} = \boldsymbol{K}_{c1}[\,\boldsymbol{R}_{w\rightarrow c1} \mid \boldsymbol{T}_{\boldsymbol{w}\rightarrow\boldsymbol{c1}}]\begin{bmatrix}X_w\\Y_w\\Z_w\\1\end{bmatrix}$$

$$\begin{bmatrix}u\\v\\w\end{bmatrix}_{c2} = \boldsymbol{K}_{\boldsymbol{c2}}[\,\boldsymbol{R}_{c2\rightarrow c1} \mid \boldsymbol{T}_{\boldsymbol{c2}\rightarrow\boldsymbol{c1}}]\begin{bmatrix}X_{c1}\\Y_{c1}\\Z_{c1}\\1\end{bmatrix} = \boldsymbol{K}_{\boldsymbol{c2}}[\,\boldsymbol{R}_{c2\rightarrow c1} \mid \boldsymbol{T}_{\boldsymbol{c2}\rightarrow\boldsymbol{c1}}]\,\boldsymbol{K}_{\boldsymbol{c1}}[\,\boldsymbol{R}_{w\rightarrow c1} \mid \boldsymbol{T}_{\boldsymbol{w}\rightarrow\boldsymbol{c1}}]\begin{bmatrix}X_w\\Y_w\\Z_w\\1\end{bmatrix}$$

Using the above to calculate the perspective matrices $\boldsymbol{P}$, and then using P to calculate $L_1 \rightarrow L_{11}$ will give you DLT coefficients that will result in the right epipolar lines. As long as you are consistent, it will also result in the correct 3D coordinates!

Use the above and DLTdv8 to get epipolar lines so you can get accurate point correspondences. However, you can also just use the extrinsic and intrinsic matrices (cv2.triangulatePoints) to get 3D

coordinates from the 2D point correspondences. You can then calculate a translations/rotation that will convert the camera1 coordinates to any arbitrary world coordinates, with the origin where you like.