Lecture 13

Camera calibration
Administrative

A3 is out
  - Due May 12th

A4 is out
  - Due May 25th
Administrative

Recitation this friday
- Ontologies
- Zihan Wang
So far: 2D Transformations with homogeneous coordinates

- No change
  \[
  \begin{bmatrix}
  1 & 0 & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 1 \\
  \end{bmatrix}
  \]
  
- Translate
  \[
  \begin{bmatrix}
  1 & 0 & X \\
  0 & 1 & Y \\
  0 & 0 & 1 \\
  \end{bmatrix}
  \]

- Scale about origin
  \[
  \begin{bmatrix}
  W & 0 & 0 \\
  0 & H & 0 \\
  0 & 0 & 1 \\
  \end{bmatrix}
  \]

- Rotate about origin
  \[
  \begin{bmatrix}
  \cos \theta & -\sin \theta & 0 \\
  \sin \theta & \cos \theta & 0 \\
  0 & 0 & 1 \\
  \end{bmatrix}
  \]

- Shear in x direction
  \[
  \begin{bmatrix}
  1 & \tan \phi & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 1 \\
  \end{bmatrix}
  \]

- Shear in y direction
  \[
  \begin{bmatrix}
  1 & 0 & 0 \\
  \tan \psi & 1 & 0 \\
  0 & 0 & 1 \\
  \end{bmatrix}
  \]

Figure: Wikipedia
For this course, we focus on the pinhole model.

- Similar to thin lens model in Physics: central rays are not deviated.
- Assumes lens camera in focus.
- Useful approximation but ignores important lens distortions.
So far: General pinhole camera matrix

\[ P = K[R|t] \quad \text{where} \quad t = -RC \]

\[
P = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_1 & r_2 & r_3 & t_1 \\ r_4 & r_5 & r_6 & t_2 \\ r_7 & r_8 & r_9 & t_3 \end{bmatrix}
\]

intrinsic parameters
extrinsic parameters
Today’s agenda

- Properties of Perspective transformations
- Introduction to Camera Calibration
- Linear camera calibration method
- Calculating intrinsics and extrinsics
- Other methods
Today’s agenda

- Properties of Perspective transformations
- Introduction to Camera Calibration
- Linear camera calibration method
- Calculating intrinsics and extrinsics
- Other methods
Similar illusion as last lecture
The Ames room illusion

- Actual position of Person A
- Apparent position of person A
- Actual and apparent position of person B
- Apparent shape of room
- Viewing peephole

Ranjay Krishna, Jieyu Zhang
Lecture 13 - 10
May 7, 2024
Projective Geometry

Q. Who is taller?
The two blue lines are the same length
Projective Geometry

What is not preserved?

- Length
- Angles
Projective Geometry

What is preserved?

- Straight lines are still straight
Projection of lines

Q. When is parallelism preserved?

Piero della Francesca, *Flagellation of Christ*, 1455-1460
Projection of lines

Q. When is parallelism preserved?
When the parallel lines are also parallel to the image plane

Piero della Francesca, *Flagellation of Christ*, 1455-1460
Projection of lines

Patterns on *non-fronto-parallel* planes are distorted by a *homography*.

---

Ranjay Krishna, Jieyu Zhang

Lecture 13 - 16

May 7, 2024
Projection of planes

What about patterns on fronto-parallel planes?

\[(x, y, z) \rightarrow \left( f \frac{x}{z}, f \frac{y}{z} \right)\]

- All points on a fronto-parallel plane are at a fixed depth \(z\)
- The pattern gets scaled by a factor of \(f / z\), but angles and ratios of lengths/areas are preserved
Vanishing Points & Lines

- **image plane**
- **vanishing point** $v$
- **camera center**
- **line in the scene**

Figure: S. Lazebnik
Vanishing Points & Lines

![Diagram of vanishing points and lines]

- Image plane
- Vanishing point \( v \)
- Camera center
- Line in the scene

Figure: S. Lazebnik
Vanishing Points & Lines
Vanishing Points & Lines

Parallel lines *in the world* intersect *in the image* at a **vanishing point**

Parallel planes *in the world* intersect *in the image* at a **vanishing line**
Vanishing lines of planes

- The projection of parallel 3D planes intersect at a vanishing line
- How can we construct the vanishing line of a plane?
Vanishing lines of planes

- The projection of parallel 3D planes intersect at a vanishing line
- How can we construct the vanishing line of a plane?
Vanishing lines of planes

**Horizon:** vanishing line of the ground plane

Q. What can the horizon tell us about the relative height of pixels with respect to the camera?
Q. Are these assertions true or false?

1. All lines that intersect in 2D are parallel in 3D
2. 2D lines always intersect each other in $P^2$
3. 3D planes always intersect each other in $P^3$
4. 3D lines always intersect each other in $P^3$
5. 3D lines always intersect each other in $P^2$
6. The projection of parallel lines in 3D meet at the same vanishing point
7. The projection of non-intersecting lines in 3D meet at the same vanishing point
8. If a set of parallel 3D lines are also parallel to a particular plane, their vanishing point will lie on the vanishing line of the plane
Q. Are these assertions true or false?

1. All lines that intersect in 2D are parallel in 3D \(\Rightarrow\) NO!
2. 2D lines always intersect each other in \(P^2\) \(\Rightarrow\) YES!
3. 3D planes always intersect each other in \(P^3\) \(\Rightarrow\) YES!
4. 3D lines always intersect each other in \(P^3\) \(\Rightarrow\) NO!
5. 3D lines always intersect each other in \(P^2\) \(\Rightarrow\) YES!
6. The projection of parallel lines in 3D meet at the same vanishing point \(\Rightarrow\) YES!
7. The projection of non-intersecting lines in 3D meet at the same vanishing point \(\Rightarrow\) NO!
8. If a set of parallel 3D lines are also parallel to a particular plane, their vanishing point will lie on the vanishing line of the plane \(\Rightarrow\) YES!
Properties of Vanishing Points & Lines

• The projections of parallel 3D lines intersect at a vanishing point
• The projection of parallel 3D planes intersect at a vanishing line
• Vanishing point <-> 3D direction of a line
• Vanishing line <-> 3D orientation of a surface

Vanishing Points are a key perspective tool:
from making realistic drawings, to measuring 3D from 2D, and even for camera calibration!
Today’s agenda

- Properties of Perspective transformations
- Introduction to Camera Calibration
- Linear camera calibration method
- Calculating intrinsics and extrinsics
- Other methods
Recap: The Pinhole Camera Model

\[ \tilde{x}_I \sim P \tilde{X}_W \]

\[ P = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & -R \end{bmatrix} \begin{bmatrix} R & -RC \\ 0 & 1 \end{bmatrix} = K[R|t] \]

**Intrinsic parameters** \( K \) (3 x 3):
- correspond to camera internals (image-to-image transformation)

**Perspective projection** (3 x 4):
- maps 3D to 2D points (camera-to-image transformation)

**Extrinsic parameters** (4 x 4):
- correspond to camera externals (world-to-camera transformation)
Camera Calibration & Pose Estimation

\[ \tilde{x}^I \sim P \tilde{X}^W \quad \text{P} = K [R \mid t] \]

How can we estimate P and its components?

**Camera Calibration**: estimating *intrinsics* K

**Pose Estimation**: estimating *extrinsics* [R \mid t]
Camera calibration from 3D-2D correspondences

Given \( n \) points with known 3D coordinates \( X_i \) and known image projections \( x_i \), estimate the camera parameters \( P \) such that \( \hat{x}_i^l \sim P \hat{X}_i^w \).
Camera calibration from 3D-2D correspondences

Given \( n \) points with known 3D coordinates \( X_i \) and known image projections \( x_i \), estimate the camera parameters \( P \) such that \( \hat{x}_i \sim P\hat{X}_i \)

<table>
<thead>
<tr>
<th>Known 2D image coords</th>
<th>Known 3D locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>880 214</td>
<td>312.747 309.140 30.086</td>
</tr>
<tr>
<td>43  203</td>
<td>305.796 311.649 30.356</td>
</tr>
<tr>
<td>270 197</td>
<td>307.694 312.358 30.418</td>
</tr>
<tr>
<td>886 347</td>
<td>310.149 307.186 29.298</td>
</tr>
<tr>
<td>745 302</td>
<td>311.937 310.105 29.216</td>
</tr>
<tr>
<td>943 128</td>
<td>311.202 307.572 30.682</td>
</tr>
<tr>
<td>476 590</td>
<td>307.106 306.876 28.660</td>
</tr>
<tr>
<td>419 214</td>
<td>309.317 312.490 30.230</td>
</tr>
<tr>
<td>317 335</td>
<td>307.435 310.151 29.318</td>
</tr>
<tr>
<td>783 521</td>
<td>308.253 306.300 28.881</td>
</tr>
<tr>
<td>235 427</td>
<td>306.500 309.301 28.905</td>
</tr>
<tr>
<td>665 429</td>
<td>308.069 306.831 29.189</td>
</tr>
<tr>
<td>655 362</td>
<td>309.671 308.834 29.029</td>
</tr>
<tr>
<td>427 333</td>
<td>308.255 309.955 29.267</td>
</tr>
<tr>
<td>412 415</td>
<td>307.546 308.613 28.963</td>
</tr>
<tr>
<td>746 351</td>
<td>311.036 309.206 28.913</td>
</tr>
<tr>
<td>434 415</td>
<td>307.518 308.175 29.069</td>
</tr>
<tr>
<td>525 234</td>
<td>309.950 311.262 29.990</td>
</tr>
<tr>
<td>716 308</td>
<td>312.160 310.772 29.080</td>
</tr>
<tr>
<td>602 187</td>
<td>311.988 312.709 30.514</td>
</tr>
</tbody>
</table>
Camera calibration from 3D-2D correspondences

Given $n$ points with known 3D coordinates $X_i$ and known image projections $x_i$, estimate the camera parameters $P$ such that $\hat{x}_i \sim P\hat{X}_i$.

Many good solutions for accurate 3D position from good fiducial markers: ArUco, AprilTags, …


[https://april.eecs.umich.edu/software/apriltag](https://april.eecs.umich.edu/software/apriltag)
Mapping between 3D point and image points

\[
\begin{bmatrix}
    x \\
    y \\
    z \\
\end{bmatrix} = \begin{bmatrix}
    p_1 & p_2 & p_3 & p_4 \\
    p_5 & p_6 & p_7 & p_8 \\
    p_9 & p_{10} & p_{11} & p_{12} \\
\end{bmatrix} \begin{bmatrix}
    X \\
    Y \\
    Z \\
    1 \\
\end{bmatrix}
\]

What are the knowns and unknowns?
Mapping between 3D point and image points

\[
\begin{bmatrix}
  x \\
  y \\
  z
\end{bmatrix} =
\begin{bmatrix}
  p_1 & p_2 & p_3 & p_4 \\
  p_5 & p_6 & p_7 & p_8 \\
  p_9 & p_{10} & p_{11} & p_{12}
\end{bmatrix}
\begin{bmatrix}
  X \\
  Y \\
  Z \\
  1
\end{bmatrix}
\]

Heterogeneous coordinates

\[
x' = \frac{p_1^\top X}{p_3^\top X} \quad y' = \frac{p_2^\top X}{p_3^\top X}
\]

(non-linear relationship between coordinates)

How can we make these relations linear?
Today’s agenda

● Properties of Perspective transformations
● Introduction to Camera Calibration
● Linear camera calibration method
● Calculating intrinsics and extrinsics
● Other methods
How can we make these relations linear?

\[
x' = \frac{p_1^\top X}{p_3^\top X} \quad y' = \frac{p_2^\top X}{p_3^\top X}
\]

Make them linear with algebraic manipulation...

\[
p_1^\top X - p_3^\top X x' = 0 \quad p_2^\top X - p_3^\top X y' = 0
\]

Now we can setup a system of linear equations with multiple corresponding points
Camera Calibration: Linear Method

\[ p_i \equiv MX_i \]

Remember (from geometry): this implies \( MX_i \) & \( p_i \) are proportional/scaled copies of each other

\[ p_i = \lambda MX_i, \lambda \neq 0 \]

Remember (from homography fitting): this implies their cross product is 0

\[ p_i \times MX_i = 0 \]
\[ p_1^T X - p_3^T X x' = 0 \]
\[ p_2^T X - p_3^T X y' = 0 \]

In matrix form …
\[
\begin{bmatrix}
  X^\top & 0 & -x' X^\top \\
  0 & X^\top & -y' X^\top \\
\end{bmatrix}
\begin{bmatrix}
p_1 \\
p_2 \\
p_3 \\
\end{bmatrix} = 0
\]

Q1. How many equations does each correspondence give us?

Q2. How many correspondences do we need to solve for P?
\[ p_1^T X - p_3^T X x' = 0 \quad \quad p_2^T X - p_3^T X y' = 0 \]

In matrix form for 1 corresponding point …
\[
\begin{bmatrix}
X^\top & 0 & -x' X^\top \\
0 & X^\top & -y' X^\top
\end{bmatrix}
\begin{bmatrix}
p_1 \\
p_2 \\
p_3
\end{bmatrix} = 0
\]

For N points …
\[
\begin{bmatrix}
X_1^T & 0 & -x'_1 X_1^T \\
0 & X_1^T & -y'_1 X_1^T \\
\vdots & \vdots & \vdots \\
X_N^T & 0 & -x'_N X_N^T \\
0 & X_N^T & -y'_N X_N^T
\end{bmatrix}
\begin{bmatrix}
p_1 \\
p_2 \\
p_3
\end{bmatrix} = 0
\]

How do we solve this system?
A few things to look out for

- N points should not be co-planar. Otherwise, the rows will not be independent.
- Usually any measurements you make in 3D space will be noisy. So you need more than the minimum number of points!
- P has 12 values but we only need to find 11 since everything is scaled.

For N points ...

\[
\begin{bmatrix}
X_1^T & 0 & -x_1'X_1^T \\
0 & X_1^T & -y_1'X_1^T \\
\vdots & \vdots & \vdots \\
X_N^T & 0 & -x_N'X_N^T \\
0 & X_N^T & -y_N'X_N^T \\
\end{bmatrix}
\begin{bmatrix}
p_1 \\
p_2 \\
p_3 \\
\end{bmatrix} = 0
\]

How do we solve this system?
Solve for camera matrix via total least squares

\[ \hat{x} = \arg \min_{x} \|Ax\|^2 \text{ subject to } \|x\|^2 = 1 \]

\[ A = \begin{bmatrix} X_1^T & 0 & -x'_1X_1^T \\ 0 & X_1^T & -y'_1X_1^T \\ \vdots & \vdots & \vdots \\ X_N^T & 0 & -x'_NX_N^T \\ 0 & X_N^T & -y'_NX_N^T \end{bmatrix} \]

\[ x = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} \]

Singular Value Decomposition!
Singular Value Decomposition (SVD)

- Represents any matrix $A$ as a product of three matrices: $U\Sigma V^T$
- Python command:
  - $[U,\Sigma,V]= numpy.linalg.svd(A)$

\[
\begin{bmatrix}
-0.40 & 0.916 \\
0.916 & 0.40
\end{bmatrix} \times 
\begin{bmatrix}
5.39 & 0 \\
0 & 3.154
\end{bmatrix} \times 
\begin{bmatrix}
-0.05 & 0.999 \\
0.999 & 0.05
\end{bmatrix} = 
\begin{bmatrix}
3 & -2 \\
1 & 5
\end{bmatrix}
\]
Singular Value Decomposition (SVD)

- Beyond 2x2 matrices:
  - In general, if $A$ is $m \times n$, then $U$ will be $m \times m$, $\Sigma$ will be $m \times n$, and $V^T$ will be $n \times n$.
  - (Note the dimensions work out to produce $m \times n$ after multiplication)

\[
U \begin{bmatrix} -.40 & .916 \\ .916 & .40 \end{bmatrix} \times \begin{bmatrix} 5.39 & 0 \\ 0 & 3.154 \end{bmatrix} \times \begin{bmatrix} -.05 & .999 \\ .999 & .05 \end{bmatrix} = \begin{bmatrix} 3 & -2 \\ 1 & 5 \end{bmatrix}
\]
Singular Value Decomposition (SVD)

- $U$ and $V$ are always **rotation** matrices.
  - Each column is a unit vector.
- $\Sigma$ is a diagonal matrix
  - The number of nonzero entries = rank of $A$
  - The algorithm always sorts the entries high to low

$$
U = \begin{bmatrix}
-.40 & .916 \\
.916 & .40 \\
\end{bmatrix} \times \begin{bmatrix}
5.39 & 0 \\
0 & 3.154 \\
\end{bmatrix} \times \begin{bmatrix}
-.05 & .999 \\
.999 & .05 \\
\end{bmatrix} = \begin{bmatrix}
3 & -2 \\
1 & 5 \\
\end{bmatrix}
$$
Solve for camera matrix via total least squares

\[ \hat{x} = \arg \min_x \| A x \|^2 \text{ subject to } \| x \|^2 = 1 \]

\[
A = \begin{bmatrix}
X_1^T & 0 & -x_1' X_1^T \\
0 & X_1^T & -y_1' X_1^T \\
\vdots & \vdots & \vdots \\
X_N^T & 0 & -x_N' X_N^T \\
0 & X_N^T & -y_N' X_N^T 
\end{bmatrix}
\]

\[
x = \begin{bmatrix}
p_1 \\
p_2 \\
p_3 
\end{bmatrix}
\]

Solution \( x \) is the column of \( V \) corresponding to the smallest singular value of \( A \).
Why is it the column of $V$ with the smallest eigenvalue?

\[ \hat{x} = \arg \min_x \|Ax\|^2 \text{ subject to } \|x\|^2 = 1 \]

Is equivalent to:

\[ x = \arg \min_x x^T(A^TA)x \text{ such that } \|x\|^2 = 1. \]
Why is it the column of V with the smallest eigenvalue?

\[ \hat{x} = \arg \min_x \|Ax\|^2 \text{ subject to } \|x\|^2 = 1 \]

Is equivalent to:

\[ x = \arg \min_x x^T (A^T A)x \quad \text{such that} \quad \|x\|^2 = 1. \]

\[
A^T A = (U \Sigma V)^T (U \Sigma V) \\
= (V^T \Sigma U^T)(U \Sigma V) \\
= V^T \Sigma (U^T U) \Sigma V \\
= V^T \Sigma I \Sigma V \\
= V^T \Sigma^2 V
\]
Why is it the column of $V$ with the smallest eigenvalue?

$$\hat{x} = \arg \min_x \|Ax\|^2 \text{ subject to } \|x\|^2 = 1$$

Is equivalent to:

$$x = \arg \min_x x^T (A^T A)x \quad \text{such that} \quad \|x\|^2 = 1.$$ 

$$A^T A = (U\Sigma V)^T (U\Sigma V)$$
$$= (V^T \Sigma U^T)(U\Sigma V)$$
$$= V^T \Sigma (U^T U) \Sigma V$$
$$= V^T \Sigma I \Sigma V$$
$$= V^T \Sigma^2 V$$

$$A^T A v_k = \sigma_k^2 v_k,$$

So, $A^T A v_k$ is proportional to the vector’s eigenvalue. So the vector corresponding to the smallest eigenvalue is what we want.
Solve for camera matrix via total least squares

\[ \hat{x} = \arg \min_x \| Ax \|^2 \text{ subject to } \| x \|^2 = 1 \]

\[
A = \begin{bmatrix}
X_1^T & 0 & -x_1'X_1^T \\
0 & X_1^T & -y_1'X_1^T \\
\vdots & \vdots & \vdots \\
X_N^T & 0 & -x_N'X_N^T \\
0 & X_N^T & -y_N'X_N^T
\end{bmatrix}
\]

\[
x = \begin{bmatrix}
p_1 \\
p_2 \\
p_3
\end{bmatrix}
\]

Equivalently, solution \( x \) is the Eigenvector corresponding to the smallest Eigenvalue of \( A \)

Ranjay Krishna, Jieyu Zhang

Lecture 13 - 50

May 7, 2024
A.2.1 Total least squares

In some problems, e.g., when performing geometric line fitting in 2D images or 3D plane fitting to point cloud data, instead of having measurement error along one particular axis, the measured points have uncertainty in all directions, which is known as the errors-in-variables model (Van Huffel and Lemmerling 2002; Matei and Meer 2006). In this case, it makes more sense to minimize a set of homogeneous squared errors of the form

\[ E_{TLS} = \sum_i (a_i x)^2 = \|A x\|^2, \]  

(A.35)

which is known as total least squares (TLS) (Van Huffel and Vandewalle 1991; Björck 1996; Golub and Van Loan 1996; Van Huffel and Lemmerling 2002).

The above error metric has a trivial minimum solution at \( x = 0 \) and is, in fact, homogeneous in \( x \). For this reason, we augment this minimization problem with the requirement that \( \|x\|^2 = 1 \). which results in the eigenvalue problem

\[ x = \arg \min_x x^T (A^T A) x \quad \text{such that} \quad \|x\|^2 = 1. \]  

(A.36)

The value of \( x \) that minimizes this constrained problem is the eigenvector associated with the smallest eigenvalue of \( A^T A \). This is the same as the last right singular vector of \( A \), because

\[ A = U \Sigma V^T, \]  

(A.37)

\[ A^T A = V \Sigma^2 V^T, \]  

(A.38)

\[ A^T A v_k = \sigma_k^2 v_k, \]  

(A.39)

which is minimized by selecting the smallest \( \sigma_k \) value.
We calculated the camera matrix!

Now we have:

\[ P = \begin{bmatrix}
    p_1 & p_2 & p_3 & p_4 \\
    p_5 & p_6 & p_7 & p_8 \\
    p_9 & p_{10} & p_{11} & p_{12}
\end{bmatrix} \]
Today’s agenda

- Properties of Perspective transformations
- Introduction to Camera Calibration
- Linear camera calibration method
- Calculating intrinsics and extrinsics
- Other methods
Recap: The Pinhole Camera Model

\[ \mathbf{\hat{x}}^I \sim \mathbf{P} \mathbf{\hat{X}}^W \]

\[
\mathbf{P} = \begin{bmatrix}
    f & 0 & p_x \\
    0 & f & p_y \\
    0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
    1 & 0 & 0 \\
    0 & 1 & 0 \\
    0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
    \mathbf{R} & -\mathbf{R} \mathbf{C} \\
    0 & 1
\end{bmatrix} = \mathbf{K} \begin{bmatrix}
    \mathbf{R} \\
    \mathbf{t}
\end{bmatrix}
\]

**intrinsic parameters** \( \mathbf{K} (3 \times 3) \) : correspond to camera internals (image-to-image transformation)

**perspective projection** (3 x 4) : maps 3D to 2D points (camera-to-image transformation)

**extrinsic parameters** (4 x 4) : correspond to camera externals (world-to-camera transformation)
Almost there … \( P = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} \)

Want to get \( P = K [R \mid -RC] \)

How can we calculate the intrinsic and extrinsic parameters from the projection matrix?
Decomposition of the Camera Matrix

\[ P = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} \sim K [ R \mid - RC ] \]

Q1. Is there a way we can get rid of \( C \)?
Decomposition of the Camera Matrix

\[ \mathbf{P} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} \sim K \begin{bmatrix} R & -RC \end{bmatrix} \]

\[ \mathbf{\bar{P}} = \begin{bmatrix} p_1 & p_2 & p_3 \\ p_5 & p_6 & p_7 \\ p_9 & p_{10} & p_{11} \end{bmatrix} \sim KR \]

Q2. Is there a way we can get rid of \( R \)?
Decomposition of the Camera Matrix

\[
P = \begin{bmatrix}
p_1 & p_2 & p_3 & p_4 \\
p_5 & p_6 & p_7 & p_8 \\
p_9 & p_{10} & p_{11} & p_{12}
\end{bmatrix} \sim K \begin{bmatrix} R & -RC \end{bmatrix}
\]

\[
\bar{P} = \begin{bmatrix}
p_1 & p_2 & p_3 \\
p_5 & p_6 & p_7 \\
p_9 & p_{10} & p_{11}
\end{bmatrix} \sim KR
\]

\[
\bar{P}\bar{P}^T \sim KR R^T K^T
\]

\[
\bar{P}\bar{P}^T \sim KK^T \quad (RR^T = I)
\]

Q. Do you remember a property of K that could help us solve this?
Decomposition of the Camera Matrix

\[
P = \begin{bmatrix}
p_1 & p_2 & p_3 & p_4 \\
p_5 & p_6 & p_7 & p_8 \\
p_9 & p_{10} & p_{11} & p_{12}
\end{bmatrix} \sim K [R | - RC]
\]

\[
\overline{P} = \begin{bmatrix}
p_1 & p_2 & p_3 \\
p_5 & p_6 & p_7 \\
p_9 & p_{10} & p_{11}
\end{bmatrix} \sim KR
\]

\[
\overline{PP}^T \sim KRR^T K^T
\]

\[
\overline{PP}^T \sim KK^T \quad (RR^T = I)
\]

K is upper triangular and positive definite!

\[
K = \begin{bmatrix}
f & 0 & p_x \\
0 & f & p_y \\
0 & 0 & 1
\end{bmatrix}
\]
Decomposition of the Camera Matrix

\[
P = \begin{bmatrix} \bar{P} \\ \begin{bmatrix} p_4 \\ p_8 \\ p_{12} \end{bmatrix} \end{bmatrix} \sim K \begin{bmatrix} R \\ -RC \end{bmatrix}
\]

\[
\bar{P}^T \bar{P} \sim K^T K \quad \text{with } K \text{ upper triangular p.d.}
\]

Obtain \( K \) by \textbf{Cholesky decomposition} of \( \bar{P}^T \bar{P} = LL^T \)

\[
K \sim L^T
\]

Scalar factor: fixed to \( 1/L_{3,3} \) (\( K_{3,3} = 1 \))

Once \( K \) is known, we can compute \( R \sim K^{-1} \bar{P} \)

Everything is calculated up to a scaling factor. So we need to rescale
Decomposition of the Camera Matrix

\[
P = \begin{bmatrix} \bar{P} & \begin{bmatrix} p_4 \\ p_8 \\ p_{12} \end{bmatrix} \end{bmatrix} \sim K [R | -RC]
\]

\[
\bar{P}^T \bar{P} \sim K^T K \quad \text{with } K \text{ upper triangular p.d.}
\]

Obtain \( K \) by \textbf{Cholesky decomposition} of \( \bar{P}^T \bar{P} = LL^T \)

\[
K \sim L^T
\]

Scalar factor: fixed to \( 1/L_{3,3} \) (\( K_{3,3} = 1 \))

Once \( K \) is known, we can compute \( R \sim K^{-1} \bar{P} \)

\( R \) is a rotation matrix: \( |R| = 1 \)!
Scaling R and calculating C

\[ |\mathbf{R}| = 1 \Rightarrow \mathbf{\lambda} = |\mathbf{K}^{-1}\mathbf{P}|^{-1/3} \]

\[ \mathbf{R} = \mathbf{\lambda} \mathbf{K}^{-1}\mathbf{P} \]

Finally, easy to know the camera center: \[ \mathbf{C} = -\mathbf{\lambda}^{-1} \mathbf{R}^T \mathbf{K}^{-1} [p_4 \ p_8 \ p_{12}]^T \]
Linear Camera Calibration

● Advantages:
  ○ Simple to formulate
  ○ Analytical solution
● Disadvantages:
  ○ Doesn’t model radial distortion (non-linear!)
  ○ Hard to impose constraints (e.g., known $f$)
  ○ Doesn’t minimize the correct error function: the reprojection error in 2D is what we truly care about!
● Hence why non-linear methods are preferred in practice.
● They can reuse the linear method we just saw!
Today’s agenda

- Properties of Perspective transformations
- Introduction to Camera Calibration
- Linear camera calibration method
- Calculating intrinsics and extrinsics
- Other methods
Non-Linear Camera Calibration

• Write down objective function in terms of intrinsic and extrinsic parameters, as sum of squared distances between measured 2D points $x_i$ and estimated projections of corresponding 3D points:

$$\sum_i \|\text{proj}(KR|t)X_i; \kappa) - x_i \|^2_2$$

• Can include radial distortion (cf. Szeliski 2.1.5 & 11.1.4) or other parameters $\kappa$ in the projection model (non-linear in the parameters!)

• Can include constraints such as known focal length, orthogonality, visibility of points, or even known $K$ (“extrinsics calibration”)

• Minimize error using standard non-linear optimization techniques (traditionally Levenberg-Marquardt, cf. Szeliski A.3, 8.1.3, 11.1.4)

• Iterative non-linear optimization is sensitive to initialization: use the output of the linear method we just saw!
Estimating depth

$b$ is the translation from camera at location $L$ and camera at $R$
Estimating depth

\[ u_L = f_x \frac{x}{z} + o_x \]

\[ v_L = f_y \frac{y}{z} + o_y \]

\[ u_R = f_x \frac{x - b}{z} + o_x \]

\[ v_R = f_y \frac{y}{z} + o_y \]
Estimating depth

\[ u_L = f_x \frac{x}{z} + o_x \]
\[ v_L = f_y \frac{y}{z} + o_y \]

\[ z = \frac{bf_x}{u_L - u_R} \]

\[ u_R = f_x \frac{x - b}{z} + o_x \]
\[ v_R = f_y \frac{y}{z} + o_y \]
Estimating depth

\[ z = \frac{bf_x}{u_L - u_R} \]
Estimating pose $[R|t]$ (extrinsics)

We have already done it together with estimating K!

What if we know K already?

Estimating $[R|t]$ only = pose estimation, a.k.a. extrinsic calibration (and previous linear method is called the “Direct Linear Transform”)

\[ p_i = (X_i, Y_i, Z_i, W_i) \]
Estimating pose $[R|t]$ (extrinsics)

Other linear algorithm: PnP (Perspective-n-Point)
Commonly used solution available in standard libraries like OpenCV
Minimal form: P3P (3 noise-free non-collinear correspondences)
Main idea: same angle between rays of 2 2D points and 2 3D points
In practice: use $n \geq 4$ correspondences + RANSAC
More details:

Quan, Long; Lan, Zhong-Dan (1999). "Linear N-Point Camera Pose Determination" (PDF). *IEEE TPAMI*.
Estimating pose $[R|t]$ (extrinsics)

Non-linear method:
- Minimize reprojection error as function of $[R|t]$ 
- More accurate and flexible (e.g., using constraints)
- Can be robustified and easy to implement via transformation decomposition and backpropagation (yes, like in Deep Learning!)

cf. Szeliski 11.2.2 for more details
Today’s agenda

● Properties of Perspective transformations
● Introduction to Camera Calibration
● Linear camera calibration method
● Calculating intrinsics and extrinsics
● Other methods
Next lecture

Recognition