Camera Calibration

COSC450

Lecture 2

Camera Calibration in OpenCV

`cv::calibrateCamera` function:

Input: Corresponding 3D and 2D points in a set of images, size of the image

Output: Calibration matrix, distortion co-efficients, camera poses, reprojection error

```
double cv::calibrateCamera(
  std::vector<std::vector<cv::Point3f>> objectPoints,
  std::vector<std::vector<cv::Point3f>> imagePoints,
  cv::Size imageSize,
  cv::Mat cameraMatrix,
  std::vector<double> distCoeffs,
  std::vector<cv::Mat> rvecs,
  std::vector<cv::Mat> tvecs);
```

Worth looking into the detail – many useful methods are used both here and elsewhere

Calibration Targets

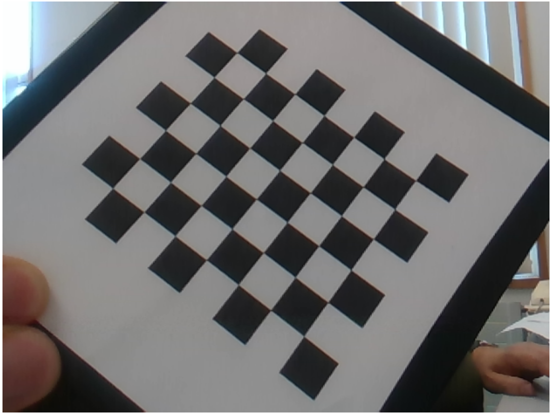
Calibration from 3D-2D matches

- Want easy to find points
- Want known 3D co-ordinates

Planar targets are common

- Easy to make with a printer
- Chess/Checkerboard patterns
- Grids of dots or lines

Is a 2D pattern enough?



3D Point Locations

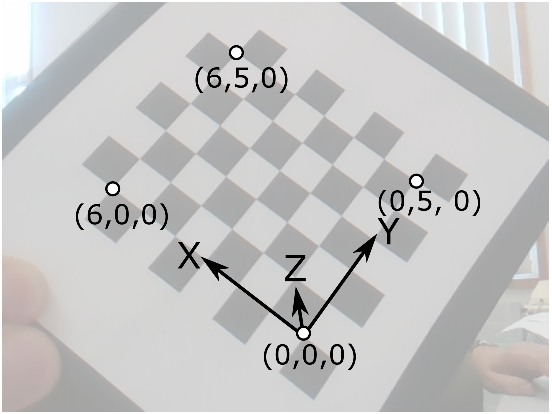
Can choose our co-ordinate frame

- $X - Y$ plane is the calibration target
- Origin is at one *internal* corner
- Z goes *into* the target

We also need to decide on units

- Best to use a real world unit
- Here squares are 1cm
- Can just use 1 square $=$ 1 unit

All points have $Z = 0$ – a problem?



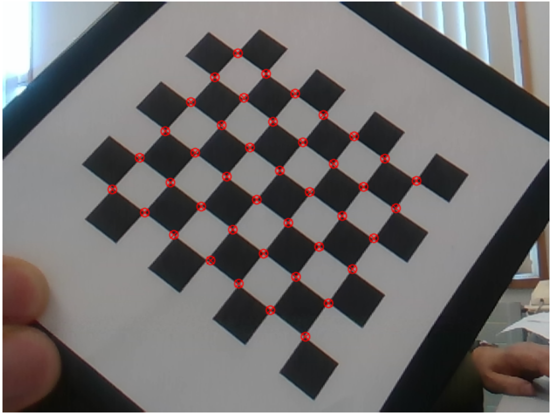
Finding Checkerboard Corners

OpenCV's method (in brief!)

- Threshold image to black & white
- Look for black & white quadrilaterals
- Link the quads into a checkerboard
- Followed by sub-pixel refinement

People are still researching this!

- Duda and Frese, BMVC 2018
- Edwards, Hayes, and Green, IVCNZ 2018
- Morten, Wilm, and Frisvad, 2019



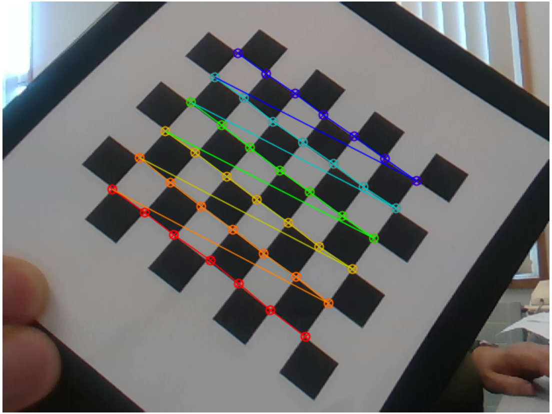
Matching 2D to 3D points

The corners are roughly in rows/cols

- OpenCV's quad-based approach helps
- Requires a view of all the corners
- Aligns 2D corners to 3D target points

Calibration targets often odd-sized

- The example here is 7×6
- Why is this helpful?
- Is this required?



The Camera Calibration Problem

For the ith image we get:

- A set of n 3D points ($j \in \{1 \dots n\}$):

$$\mathbf{x}_{i,j} = \begin{bmatrix} x_{i,j} & y_{i,j} & z_{i,j} & 1 \end{bmatrix}^\mathsf{T}$$

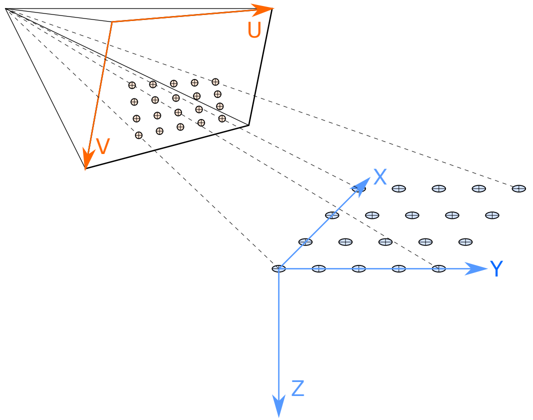
- Corresponding 2D points:

$$\mathbf{u}_{i,j} = \begin{bmatrix} u_{i,j} & v_{i,j} & 1 \end{bmatrix}^\mathsf{T}$$

- Related by

$$\mathbf{u}_{i,j} \equiv \mathrm{K} \begin{bmatrix} \mathrm{R}_i & \mathbf{t}_i \end{bmatrix} \mathbf{v}_{i,j}$$

Want to find K (get R_is and $\mathbf{t}_i$s as a bonus)



Zhang's Calibration Method – Notation

450	Zhang	Description
$\mathbf{x}$	$\tilde{\mathbf{M}}$	3D homogeneous point
$\mathbf{u}$	$\tilde{\mathbf{m}}$	2D homogeneous point
$\tilde{\mathbf{x}}$	$\tilde{\mathbf{M}}$	Planar homogenous point
R	$\mathbf{R}$	Camera rotation matrix
$\mathbf{r}_i$	$\mathbf{r}_i$	ith column of R
$\mathbf{t}$	$\mathbf{t}$	Camera translation vector

450	Zhang	Description
K	$\mathbf{A}$	Calibration matrix
f_u	α	Focal length in u
f_v	β	Focal length in v
s	γ	Camera pixel skew
c_u	u_0	Principal point u
c_v	v_0	Principal point v
–	s	Homogeneous scale factor

Method Summary

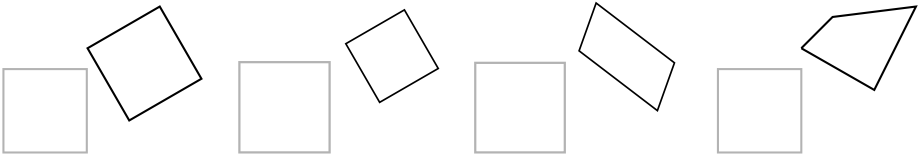
Basic steps:

- Find a *homography* between $\mathbf{x}$s and $\mathbf{u}$s
- Use this to find constraints on K
- Solve for and estimate of K
  - We get Rs and $\mathbf{t}$s as well
- (Optional) add in lens distortions
- Refine estimate by minimising reprojection error

Uses several common techniques

- Homographies
- Solving linear systems of equations
- Reprojection error
- Non-linear least squares

Transformations in 2D



	Euclidean	**Similarity**	**Affine**	**Homography**
ma DoF	3	4	6	8
Matrix	$\begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} sr_{11} & sr_{12} & t_x \\ sr_{21} & sr_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ a_{21} & a_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix}$
Preserves	Length Area	Ratios of lengths Angles	Parallelism Ratios of areas	Co-linearity Ordering on a line

Finding a Homography

We start with the projection equation

$$\mathbf{u} \equiv \mathrm{K} \begin{bmatrix} \mathrm{R} & \mathbf{t} \end{bmatrix} \mathbf{x}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \begin{bmatrix} f_u & s & c_u \\ 0 & f_v & c_v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 & \mathbf{t} \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} f_u & s & c_u \\ 0 & f_v & c_v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \vdots & \vdots & \vdots \\ \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t} \\ \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\mathbf{u} \equiv \mathrm{K} \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t} \end{bmatrix} \tilde{\mathbf{x}}$$

This defines a *homography* $\mathbf{u} \equiv \mathrm{H}\tilde{\mathbf{x}}$

- Mapping between two images of a plane
- Defined by a 3×3 matrix, H
- Only defined up to a scale – a *homogeneous* quantity
- General linear transform
- Preserves straight lines
- Also used in panaoramic image stitching

Algorithm Overview

Each image gives us a homography, H

- The homography has 9 values
  - Only defined up to a scale
  - Only 8 *independent* values
  - 8 *degrees of freedom*
- Six are the pose of the camera
  - Three for translation
  - Three for rotation
- Leaving us two constraints on K
- So we need at least three images

So the overall algorithm is for each image:

1. Find the homography H
2. Derive two constraints on K

Next, using all the images' constraints

3. Estimate the value of K

Finally,

4. Estimate R and $\mathbf{t}$ for each image
5. Refine the estimate, adding in lens distortion parameters

1 – Finding the Homography

The Direct Linear Transform

For a 2D–3D match, $\mathbf{u} \equiv \mathrm{H}\tilde{\mathbf{x}}$ so $\mathbf{u}$ is parallel to $\mathrm{H}\tilde{\mathbf{x}}$

$$\mathbf{u} \times \mathrm{H}\tilde{\mathbf{x}} = \mathbf{0}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \times \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & -x & -y & -1 & xv & yv & v \\ x & y & 1 & 0 & 0 & 0 & -xu & -yu & -u \\ -xv & -yv & -v & xu & yu & u & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ \vdots \\ h_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

The Direct Linear Transform

This gives us three equations in H

- They are linear – which is good
- They are not independent – use first two

$$\begin{bmatrix} \mathbf{0}^\top & -\tilde{\mathbf{x}}^\top & v\tilde{\mathbf{x}}^\top \end{bmatrix} \mathbf{h} = 0$$
$$\begin{bmatrix} \tilde{\mathbf{x}}^\top & \mathbf{0}^\top & -u\tilde{\mathbf{x}}^\top \end{bmatrix} \mathbf{h} = 0$$

- H has eight degrees of freedom (why?)
- So need at least four points

For n points we get the system

$$\begin{bmatrix} \mathbf{0}^\top & -\tilde{\mathbf{x}}_1^\top & v_1\tilde{\mathbf{x}}_1^\top \\ \tilde{\mathbf{x}}_1^\top & \mathbf{0}^\top & -u_1\tilde{\mathbf{x}}_1^\top \\ \mathbf{0}^\top & -\tilde{\mathbf{x}}_2^\top & v_2\tilde{\mathbf{x}}_2^\top \\ \tilde{\mathbf{x}}_2^\top & \mathbf{0}^\top & -u_2\tilde{\mathbf{x}}_2^\top \\ \vdots & \vdots & \vdots \\ \mathbf{0}^\top & -\tilde{\mathbf{x}}_n^\top & v_n\tilde{\mathbf{x}}_n^\top \\ \tilde{\mathbf{x}}_n^\top & \mathbf{0}^\top & -u_n\tilde{\mathbf{x}}_n^\top \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ \vdots \\ h_{32} \\ h_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

$$\mathrm{A}\mathbf{h} = \mathbf{0}$$

Linear Systems of Equations

Simple case:

- We have n unknowns, $x_1, x_2, \ldots x_n$
- We have m equations of the form

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \ldots a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + \ldots a_{2n}x_n &= b_2 \\
\vdots \quad \vdots \quad \vdots \quad & \vdots \quad \vdots \\
a_{m1}x_1 + a_{m2}x_2 + \ldots a_{mn}x_n &= b_m
\end{aligned}
$$

Writing this more compactly,

$$\mathrm{A}\mathbf{x} = \mathbf{b}$$

- A is an $m \times n$ matrix

$$\mathrm{A} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix}$$

- $\mathbf{x}$ is a vector of n unknowns

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \ldots & x_n \end{bmatrix}^\mathsf{T}$$

- $\mathbf{b}$ is a vector of m known constants

$$\mathbf{b} = \begin{bmatrix} b_1 & b_2 & \ldots & b_m \end{bmatrix}^\mathsf{T}$$

Solving Linear Systems

Simple case:

- A is square so $m = n$
- All equations are independent...
- ...so A is invertible
- At least one $b_i \neq 0$, so $\mathbf{b} \neq \mathbf{0}$

Solution is usually given as:

$$\mathbf{x} = \mathrm{A}^{-1}\mathbf{b}$$

Complicated cases:

- More equations than unknowns, $m > n$
- Fewer equations than unknowns, $m < n$
- Homogeneous system: $\mathbf{b} = \mathbf{b}$

Solutions use the singular value decomposition (SVD)

- First case gives least squares fit
- Other two give families of solutions
- SVD is also more stable for simple case

The Singular Value Decomposition

Any real $m \times n$ matrix, A can be written as

$$\mathrm{A} = \mathrm{USV}^{\mathsf{T}}$$

- U is an $m \times m$ *orthonormal* matrix
- S is an $m \times n$ diagonal matrix
- V is an $n \times n$ *orthonormal* matrix

Orthonormal matrices:

- Every row/column is a unit vector
- Different rows/columns are orthogonal
- Transpose is inverse, $M^{-1} = M^{\mathsf{T}}$

Simple case $\mathrm{A}\mathbf{x} = \mathbf{b}$

$$\begin{aligned} \mathbf{x} &= \mathrm{A}^{-1}\mathbf{b} \\ &= (\mathrm{USV}^{\mathsf{T}})^{-1}\mathbf{b} \\ &= \mathrm{VS}^{-1}\mathrm{U}^{\mathsf{T}}\mathbf{b} \end{aligned}$$

- S is easy to invert:

$$\begin{bmatrix} s_1 & 0 & \dots & 0 \\ 0 & s_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & s_n \end{bmatrix}^{-1} = \begin{bmatrix} \frac{1}{s_1} & 0 & \dots & 0 \\ 0 & \frac{1}{s_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{1}{s_n} \end{bmatrix}$$

Solving Non-Square Linear Systems

More equations than unknowns, $m > n$

- A is not square, so no inverse
- Can form a *pseudoinverse*

$$\mathrm{A}^{+} = \mathrm{V}\mathrm{S}^{+}\mathrm{U}^{\mathsf{T}},$$

- S^{+} is an $n \times m$ diagonal matrix

$$\mathrm{S}^{+} = \begin{bmatrix} \frac{1}{s_1} & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & \frac{1}{s_2} & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & & & \\ 0 & 0 & \dots & \frac{1}{s_n} & 0 & \dots & 0 \end{bmatrix}$$

- Least-squares fit: $\mathbf{x} = \mathrm{A}^{+}\mathbf{b}$

Fewer equations than unknowns, $m < n$

- Again, A is not square
- S has columns of zeros:

$$\mathrm{S} = \begin{bmatrix} s_1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & s_2 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & & & \\ 0 & 0 & \dots & s_n & 0 & \dots & 0 \end{bmatrix}$$

- Many solutions – corresponding columns of V span the solution space
- Smallest solution (minimising $\mathbf{x}^{\mathsf{T}}\mathbf{x}$) is

$$\mathbf{x} = \mathrm{A}^{+}\mathbf{b}$$

Solving Homogeneous Equations

Our problem is $\mathrm{A}\mathbf{h} = \mathbf{0}$

- Trivial solution: $\mathbf{h} = \mathbf{0}$
- More interesting solutions

$$\mathrm{A}\mathbf{h} = \mathbf{0} = 0\mathbf{h}$$

- This is an *eigenvector* of A
- The corresponding *eigenvalue* is 0
- Eigenvalues and the SVD are closely related

Eigenvalues and SVD, $\mathrm{A} = \mathrm{USV}^\mathsf{T}$:

- Columns of U are eigenvectors of AA^T
- Columns of V are eigenvectors of $\mathrm{A}^\mathsf{T}\mathrm{A}$
- S gives square roots of eigenvalues

$$\mathrm{A}^\mathsf{T}\mathrm{A}\mathbf{h} = \mathrm{A}^\mathsf{T}\mathbf{0} = \mathbf{0}$$

,

- Zeroes on diagonal of $S \rightarrow$ solutions
- These are corresponding columns of V

Normalising Transforms

Now we can find **h**

- Form the matrix A
- Take the SVD and find the eigenvector for the smallest (≈ 0) diagonal of S

This doesn't work well in practice

- Entries of A vary in size
- u and v are typically ~ 1000
- x and y depend on world units
- Changing some entries give large effects
- Others need large changes to correct

We apply normalising transforms

$$\mathbf{u}' = \mathrm{T}_u \mathbf{u} \qquad \tilde{\mathbf{x}}' = \mathrm{T}_x \tilde{\mathbf{x}}$$

- T_u and T_x are translation + scale
- Means of $\mathbf{u}'$ and $\tilde{\mathbf{x}}'$ are zero
- Means of $\|\mathbf{u}'\|$ and $\|\tilde{\mathbf{x}}'\|$ are $\sqrt{2}$
- Find homography H' so that $\mathbf{u}' = \mathrm{H}'\tilde{\mathbf{x}}'$
- Then $\mathrm{H} = \mathrm{T}_u^{-1}\mathrm{H}'\mathrm{T}_x$

Algorithm – Computing a Homography

procedure FindHomography($n \geq 4$ matches $\mathbf{u}_i \leftrightarrow \mathbf{x}_i$)

Compute normalising transforms T_u and T_x

Form a $2n \times 9$ matrix A

for i in $\{1 \dots n\}$ **do**

Compute $\mathbf{u}'_i = [u'_i \; v'_i \; 1]^\mathsf{T} \equiv \mathrm{T}_u \mathbf{u}_i$ and $\mathbf{x}'_i = [x'_i \; y'_i \; 1]^\mathsf{T} \equiv \mathrm{T}_x \mathbf{x}_i$

$\mathrm{A}_{2i,:} \leftarrow \begin{bmatrix} 0 & 0 & 0 & -x'_i & -y'_i & -1 & v'_i x'_i & v'_i y'_i & v'_i \end{bmatrix}$

$\mathrm{A}_{2i+1,:} \leftarrow \begin{bmatrix} x'_i & y'_i & 1 & 0 & 0 & 0 & -u'_i x'_i & -u'_i y'_i & -u'_i \end{bmatrix}$

end for

Compute the SVD $\mathrm{A} = \mathrm{USV}^\mathsf{T}$

Find $\mathbf{h}'$ as the column of V corresponding to smallest entry in S

$\mathrm{H}' \leftarrow \mathbf{h}'$ reshaped to a 3×3 matrix

return $\mathrm{H} = \mathrm{T}_u^{-1} \mathrm{H}' \mathrm{T}_x$

end procedure

2 – Deriving Constraints on K

Constraints from the Homography H

Recall that $\mathrm{H} \equiv \mathrm{K} \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t} \end{bmatrix}$

- Columns of H are

$$\mathbf{h}_1 = \lambda \mathrm{K} \mathbf{r}_1 \quad \mathbf{h}_2 = \lambda \mathrm{K} \mathbf{r}_2 \quad \mathbf{h}_3 = \lambda \mathrm{K} \mathbf{t}$$

- λ is a unknown scale factor
- $\mathbf{r}_1$ and $\mathbf{r}_2$ are unit vectors
- $\mathbf{r}_1$ and $\mathbf{r}_2$ are orthogonal

This gives us two constraints on K from H

$$\mathbf{r}_1^\mathsf{T} \mathbf{r_2} = \mathbf{h}_1 \mathrm{K}^{-\mathsf{T}} \mathrm{K}^{-1} \mathbf{h}_2$$
$$\mathbf{h}_1 \mathrm{K}^{-\mathsf{T}} \mathrm{K}^{-1} \mathbf{h}_1 = \mathbf{h}_2 \mathrm{K}^{-\mathsf{T}} \mathrm{K}^{-1} \mathbf{h}_2$$

We know H, so the constraints are on

$$\mathrm{B} = \mathrm{K}^{-\mathsf{T}} \mathrm{K}^{\mathsf{T}}$$

This matrix B is symmetric

$$\mathrm{B} = \begin{bmatrix} b_1 & b_2 & b_3 \\ b_2 & b_4 & b_5 \\ b_3 & b_5 & b_6 \end{bmatrix}$$

And our constraints are of the form

$$\mathbf{h}_i^\mathsf{T} \mathrm{B} \mathbf{h}_j$$

3 – Estimating K

Using the Constraints

We can re-write the constraints in the form $\mathbf{v}_{ij}^{\mathsf{T}}\mathbf{b}$, where

$$\mathbf{b} = \begin{bmatrix} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 \end{bmatrix}^{\mathsf{T}}$$
$$\mathbf{v}_{ij} = \begin{bmatrix} h_{i1}h_{j1} & h_{i1}h_{j2} + h_{i2}h_{j1} & h_{i1}h_{j3} + h_{i3}h_{j1} & h_{i2}h_{j2} & h_{i2}h_{j3} + h_{i3}h_{j2} & h_{i3}h_{j3} \end{bmatrix}^{\mathsf{T}}$$

Each calibration image gives us a homography, so two constraints on B

$$\mathbf{v}_{12}^{\mathsf{T}}\mathbf{b} = 0 \qquad (\mathbf{v}_{11}^{\mathsf{T}} - \mathbf{v}_{22}^{\mathsf{T}})\mathbf{b} = 0$$

Three (or more) images gives us six (or more) equations, so solve for **b** as for **h**

Can then recover K from B (see paper for details)

4 – Estimating R and $\mathbf{t}$ for each Image

Wait - Aren't We Done?

We solved for H then K

- Used a linear solution
- These are easy to solve

Typically we have more points than needed

- More than 4 points on the pattern
- More than 3 images of the pattern

This lets us minimise the effects of errors

- Errors in measurements
- What function have we minimised?

Reprojection error

- We *measure* $\mathbf{u}$ in an image
- We *estimate* $\tilde{\mathbf{u}} = \mathrm{K}\begin{bmatrix}\mathrm{R} & \mathbf{t}\end{bmatrix}\mathbf{x}$
- Reprojection error is $\mathbf{u} - \tilde{\mathbf{u}}$

If we have n images with m points

- Want to minimise total error
- Usually sum of squared errors
- Optimal assuming Gaussian errors

$$\epsilon = \sum_{i=1}^{n}\sum_{j=1}^{m}\|\mathbf{u}_{ij} - \tilde{\mathbf{u}}_{ij}\|^2$$

Decomposing the Homography

We've estimated H

- We know $\mathrm{H} \equiv \mathrm{K}\begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t}\end{bmatrix}$

$$\mathrm{H} = \begin{bmatrix}\mathbf{h}_1 & \mathbf{h}_2 & \mathbf{h}_3\end{bmatrix} = \lambda\mathrm{K}\begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t}\end{bmatrix}$$

- We've also estimated K via B
- Can then solve

$$\mathbf{r}_1 = \lambda\mathrm{K}^{-1}\mathbf{h}_1 \quad \mathbf{r}_2 = \lambda\mathrm{K}^{-1}\mathbf{h}_2 \quad \mathbf{t} = \lambda\mathrm{K}^{-1}\mathbf{h}_3$$

- We know $\|\mathbf{r}_1\| = \|\mathbf{r}_2\| = 1$, so

$$\lambda = \frac{1}{\|\mathrm{K}^{-1}\mathbf{h}_1\|} = \frac{1}{\|\mathrm{K}^{-1}\mathbf{h}_2\|}$$

This gives us $\mathbf{t}$ and most of R

- The third column of R is

$$\mathbf{r}_3 = \mathbf{r}_1 \times \mathbf{r}_2$$

- R is not a 'proper' rotation
- This is due to estimation errors
- If we take the SVD

$$\mathrm{R} = \mathrm{USV}^\mathsf{T}$$

then the nearest true rotation is

$$\mathrm{R} = \mathrm{UIV}^\mathsf{T}$$

5 – Refining the Estimate

Non-Linear Least Squares

We know have *estimates* of:

- The calibration matrix K
- The rotation and translation

$$\mathrm{R}_i \qquad \mathbf{t}_i$$

  for each image $i = 1 \ldots n$
- The 3D location

$$\mathbf{x}_j = \begin{bmatrix} x_j & y_j & z_j & 1 \end{bmatrix}^\mathsf{T}$$

  for each 3D point $j = 1 \ldots m$
- The 2D image points $\mathbf{u}_{ij}$

We want to minimise

$$\begin{aligned} \epsilon &= \sum_{i=1}^{n} \sum_{j=1}^{m} \|\mathbf{u}_{ij} - \tilde{\mathbf{u}}_{ij}\|^2 \\ &= \sum_{i=1}^{n} \sum_{j=1}^{m} \|\mathbf{u}_{ij} - \mathrm{K} \begin{bmatrix} \mathrm{R} & \mathbf{t} \end{bmatrix} \mathbf{x}\|^2 \end{aligned}$$

- This is a sum of squared terms
- But the terms are non-linear
- This makes minimising it hard

Gradient Descent

Suppose we have an error function, $f(x)$

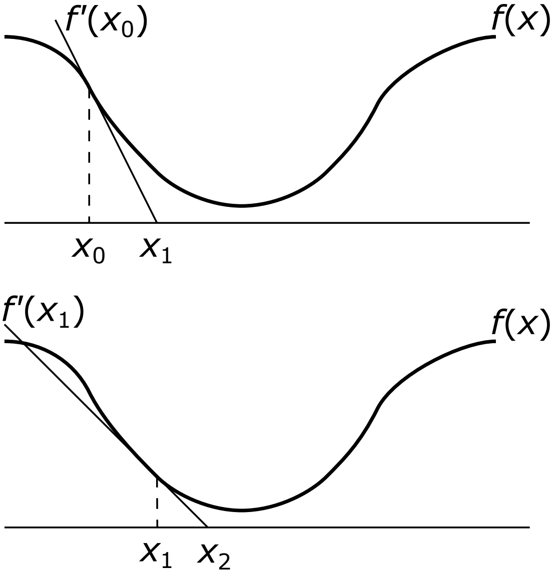
- We want to find x to minimise f
- We have an initial estimate, x_o
- We make a linear approximation

$$f(x_0 + \delta) \approx f(x_0) + \delta f'(x_0)$$

- We update $x_{i+1} \leftarrow x_i + \delta$ to reduce f
- Our step is down the gradient

$$\delta = -\lambda f'(x_i)$$

- A small enough $\lambda > 0$ always helps



Gradient Descent in Higher Dimensions

The same basic approach applies

- The parameters become a vector
- The function can be vector valued

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1\left(\begin{bmatrix} x_1 & x_2 & \dots & x_k \end{bmatrix}^\mathsf{T}\right) \\ f_2\left(\begin{bmatrix} x_1 & x_2 & \dots & x_k \end{bmatrix}^\mathsf{T}\right) \\ \vdots \\ f_n\left(\begin{bmatrix} x_1 & x_2 & \dots & x_k \end{bmatrix}^\mathsf{T}\right) \end{bmatrix}$$

- The derivatives are the *Jacobian* matrix

$$\mathrm{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_k} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_k} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_k} \end{bmatrix}$$

Linear approximation becomes

$$\mathbf{f}(\mathbf{x}_0 + \boldsymbol{\delta}) \approx \mathbf{f}(\mathbf{x}_0) + \mathrm{J}(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0)$$

Gradient descent update is

$$\boldsymbol{\delta} = -\lambda \mathrm{J}^\mathsf{T}\mathbf{f}(\mathbf{x}_i)$$

Levenberg-Marquardt

Gradient descent

- Small enough steps always help
- Can be slow to converge

Faster convergence with *Gauss–Newton*

- Update is the *normal equations*

$$\mathrm{J}^{\mathsf{T}}\mathrm{J}\boldsymbol{\delta} = -\mathrm{J}^{\mathsf{T}}\mathbf{f}(\mathbf{x}_0)$$

- Faster to converge
- May not always improve

Levenberg-Marquardt algorithm:

- Update equation is

$$\left(\mathrm{J}^{\mathsf{T}}\mathrm{J} + \lambda\mathrm{I}\right)\boldsymbol{\delta} = -\mathrm{J}^{\mathsf{T}}\mathbf{f}(\mathbf{x}_0)$$

- If a step improves the error:
  - Accept the correction
  - Reduce λ and take another step
  - This moves towards Gauss-Newton
- If the step makes things worse
  - Reject the correction
  - Increase λ and try again
  - This moves towards gradient descent

Algorithm – Levenberg-Marquardt (Basic Version)

procedure $\textsc{LevenbergMarquardt}$(function $\mathbf{f}$, parameter estimate $\mathbf{p}$)
  Compute J at $\mathbf{p}$
  Initialise $\lambda = 0.001$ ▷ Or similar small value
  while not done **do**
    Solve $(\mathrm{J}^\mathsf{T}\mathrm{J} + \lambda \mathrm{I})\boldsymbol{\delta} = \mathrm{J}^\mathsf{T}\mathbf{f}(\mathbf{p})$
    $\mathbf{p}' \leftarrow \mathbf{p} + \boldsymbol{\delta}$
    if $\mathbf{f}(\mathbf{p}') < \mathbf{f}(\mathbf{p})$ **then** ▷ This step has helped
      $\mathbf{p} \leftarrow \mathbf{p}'$ ▷ Accept the update
      Recompute J at new $\mathbf{p}$
      $\lambda \leftarrow \lambda/2$ ▷ Move towards Gauss-Newton
    else
      $\lambda \leftarrow \lambda \times 2$ ▷ Move towards Gradient Descent
    end if
  end while
  return p
end procedure

Modelling Lens Distortion

- Barrel/pincushion distortion

$$\mathbf{u}' = \mathbf{u}(1 + k_1 r^2 + k_2 r^4 + k_3 r^6 + \dots)$$

  - (u, v) measured from image centre
  - r is distance from image centre
- Tangential distortion

$$u' = u + (2p_1 uv + p_2(r^2 + 2u^2))$$
$$v' = v + (2p_2 uv + p_1(r^2 + 2v^2))$$

  - Lens not parallel to image plane
- Add k_1, k_2, k_3, p_1, p_2 to estimation