COMPUTER VISION 02504

3D aspects of computer vision:

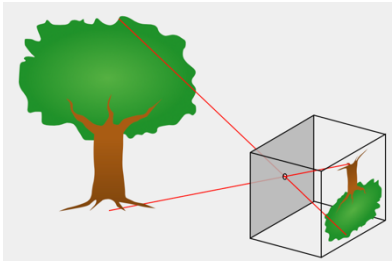
- Camera model
- Multi view geometry
- 3D reconstruction
- Image features and matching

Week 1: Camera Geometry

Pinhole Camera

What is a "good" camera model?
In terms of accuracy vs. ease-of-use?

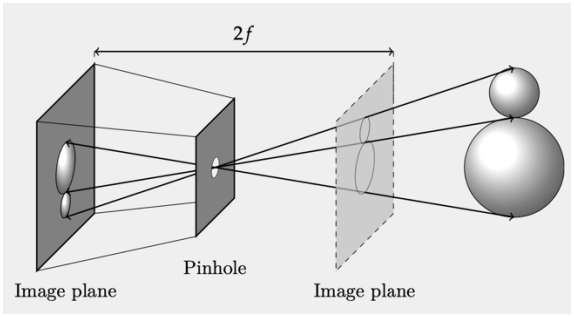
1. As accurate as possible
2. As easy to use as possible
3. Somewhere between 1 and 2



The projected image appears upside down.
Each point in an image corresponds to a direction from the camera.

- A point seen in a single camera must be along a specific line in the other camera.
- Seeing same point in two cameras is enough to find the point in 3D.

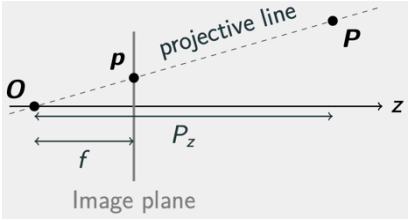
Perspective Transformations



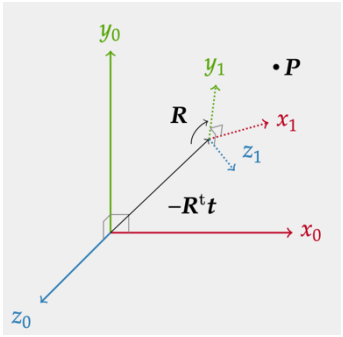
When modelling we place the "image plane" in front of the camera. The distance form image plane to camera is the *focal length f.*

Perspective projection:

$$P = \begin{pmatrix} P_x \\ P_y \\ P_z \end{pmatrix}, \quad p = \frac{f}{P_z}\begin{pmatrix} P_x \\ P_y \end{pmatrix}$$



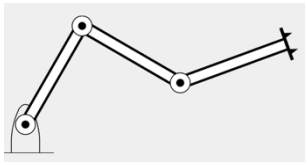
Euclidean (rigid) Transformations



Rotations and translations:
Rotation matrix R and translation vector t:

$$P_1 = R_1 P_0 + t_1$$

Ex: Robot arm transformation



$$P_1 = R_1 P_0 + t_1$$
$$P_2 = R_2 P_1 + t_2$$
$$P_3 = R_3 P_2 + t_3$$
$$P_4 = R_4 P_3 + t_4$$

$$P_4 = R_4(R_3(R_2(R_1 P_0 + t_1) + t_2) + t_3) + t_4$$
$$P_4 = R_4 R_3 R_2 R_1 P_0 + R_4 R_3 R_2 t_1 + R_4 t_3 + R_4 R_3 t_2 + t_4$$

Homogeneous coordinates

We can use 4 numbers to represent numbers in 3D as a fourth coordinate.

$P = \begin{pmatrix} P_x \\ P_y \\ P_z \end{pmatrix}$ 3D point

$P = \begin{pmatrix} s\,P_x \\ s\,P_y \\ s\,P_z \\ s \end{pmatrix}$ Adding a four coordinate

Euclidian transformation:
Rotation $R = [r_1 \; r_2 \; r_3]$, with columns r_i, and translation t.

$$P_1 = RP_0 + t = r_1 P_x + r_2 P_y + r_3 P_z + t$$

$$P_1 = \underbrace{[r_1 \quad r_2 \quad r_3 \quad t]}_{\text{Transform } \tilde{T}} \underbrace{\begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix}}_{\text{Homogeneous } P_{0h}} = \tilde{T}\, P_{0h}$$

Fully homogeneous Euclidean transformations become:

$$P_{1h} = T\, P_{0h}$$
$$\begin{bmatrix} P_1 \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} P_0 \\ 1 \end{bmatrix}$$

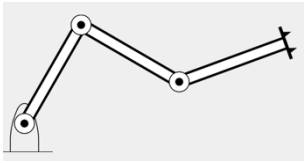
The homogeneous transformation T takes on the 4x4 form:

$$T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$$

Rotation and translation in 3D by a single matrix multiplication.

Ex: Robot arm and homogeneous transformations
Not just easier, but computationally faster.

$$Q_{4h} = T_4 T_3 T_2 T_1 Q_{0h}$$

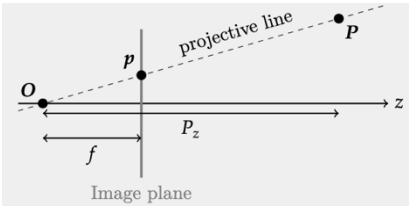


So, we can compute the position of the end point by multiplying four times or by a single 4x4 matrix.

The inhomogeneous p corresponds to the homogeneous p_h by $p_h = \begin{bmatrix} s\,p \\ s \end{bmatrix}$

The projective transformation:

$$p_h = \begin{bmatrix} s\,p \\ s \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ s \end{bmatrix} = \begin{bmatrix} fP_x/P_z \\ fP_y/P_z \\ s \end{bmatrix}$$



$$P = \begin{pmatrix} P_x \\ P_y \\ P_z \end{pmatrix}$$

Projecting 3D points:

Let us assume f=1 for simplicity: $P = \begin{pmatrix} P_x \\ P_y \\ P_z \end{pmatrix} = P_z\, p_h$

Projective transformation is like assuming point in 3D is a 2D homogeneous point.

Any homogeneous coordinate q corresponds to the same inhomogeneous p by $q = \begin{bmatrix} s\,p \\ s \end{bmatrix}$

Lines in homogeneous coordinates:

$$0 = ax + by + c$$

If we have a point in homogeneous coordinates:

$$= \begin{pmatrix} a \\ b \\ c \end{pmatrix} \cdot \begin{pmatrix} sx \\ sy \\ s \end{pmatrix} = I \cdot p_h$$

If $a^2 + b^2 = 1$ then $I \cdot p_h$ yields the closest (signed) distance from the point to the line.

Summary of homogeneous coordinates

The additional imaginary scale s, or alternatively dimension w

$$u = s \begin{pmatrix} v \\ 1 \end{pmatrix} = \begin{pmatrix} sv \\ s \end{pmatrix} = \begin{pmatrix} v' \\ w \end{pmatrix}$$

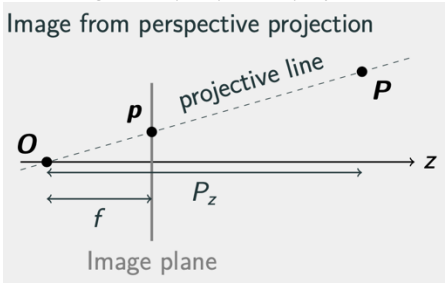
- Dimensionality is $N + 1$
- Points have a scale $s \neq 0$
- Directions have $w = 0$
- Many-to-one correspondence: $u \in \mathbb{R}^{N+1}$ and $v \in \mathbb{R}^N$

Getting inhomogeneous coordinates back:

$$v = \Pi(u) = \Pi \begin{pmatrix} v' \\ s \end{pmatrix} = v'/s \text{ (Trivial inverse)}$$

Pinhole camera model

Where is (0,0) in these images?

Image on your computer	Image from perspective projection
	
- Upper left corner	- Optical center (approximately in the middle of the image)

We introduce δ_x and δ_y to move these points to the same place. This is called **the principal point.**

Principal point:
Projection is now

$$p_x = \frac{f}{P_z} P_x + \delta_x \,, \quad p_y = \frac{f}{P_z} P_y + \delta_y$$

Intrinsics

Can we write all of this using homogeneous coordinates?

$$p_h = \begin{bmatrix} f & 0 & \delta_x \\ 0 & f & \delta_y \\ 0 & 0 & 1 \end{bmatrix} P = k\, P$$

This is called the camera matrix. It contains intrinsic camera parameters.

Extrinsics

- The extrinsics of the camera are the rotation (R) and translation (t).
- They describe the pose of the camera.
- To project points, we first transform them to the reference frame of the camera:

$$P_{cam} = R\, P + t = [R \quad t] \begin{bmatrix} P \\ 1 \end{bmatrix}$$

Projection matrix:
Projecting a single point in 3D to the camera:

$$p_h = K\, P_{cam} = K\, [R \quad t] \quad P_h$$

$$\begin{bmatrix} s\, p_x \\ s\, p_y \\ s \end{bmatrix} = \begin{bmatrix} f & 0 & \delta_x \\ 0 & f & \delta_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix}$$

- The translation is not the position of the camera.
- The final coordinate of P_h must be 1.

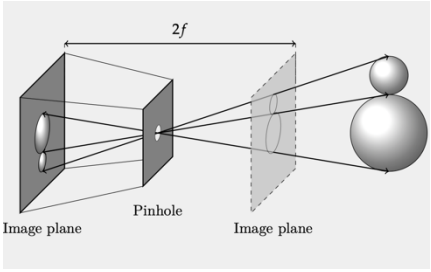
Projection matrix:

$$q = P\, Q = K\, [R \quad t]\, Q$$

The matrix P is known as the projection matrix (don't call it the camera matrix).

Week 2: Camera model and homographies

Pinhole camera



$$K = \begin{bmatrix} f & 0 & \delta_x \\ 0 & f & \delta_y \\ 0 & 0 & 1 \end{bmatrix}$$

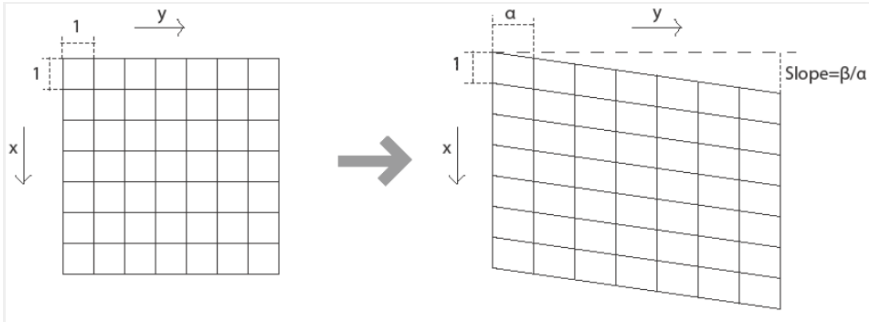
Camera Intrinsics

- Focal length f: distance from the pinhole to the image plane in pixels.
  This corresponds to the zoom level/field of view.
  Longer focal length = more zoomed in.

Dolly zoom/vertigo effect What does the focal length do? Moving camera closer while zooming out to make one object have a constant size is a popular effect.

- Principal point δ_x, δ_y
- Skew parameters α and β: Non-square pixels (α) and non-rectangular pixels (β).

$$K = \begin{bmatrix} f & \beta f & \delta_x \\ 0 & \alpha f & \delta_y \\ 0 & 0 & 1 \end{bmatrix}$$



Orthographic projection:
If all projection lines are parallel, then magnification m=1.

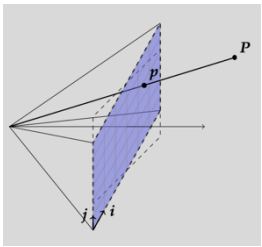
$$p = \begin{pmatrix} P_x \\ P_y \end{pmatrix}$$

The pinhole model towards orthographic projection as the focal length goes to ∞.

Projection matrix: Degrees of freedom

$$P = K\,R\,t = \begin{bmatrix} f & \beta f & \delta_x \\ 0 & \alpha f & \delta_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}$$

P is 3 x 4 = 12 numbers.
5 + 3+ 3 + 1 (scale) = 12 degrees of freedom.



Which physical properties do I need to derive the intrinsics: $f, \delta_x, \delta_x, \alpha, \beta$?

Speculate: Given an unknown system, how would you estimate all parameters in P ?

Lens distortion

Real lenses

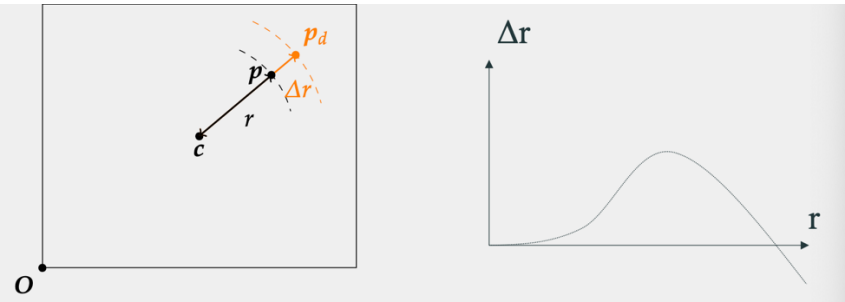
Lenses are:

- Complicated by many effects, intentional or non-intentional
- Imperfect with, for example, defects
- Different for each camera/lens

Almost all lenses have lens distortion!

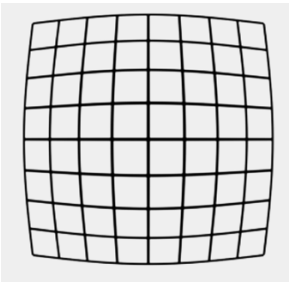
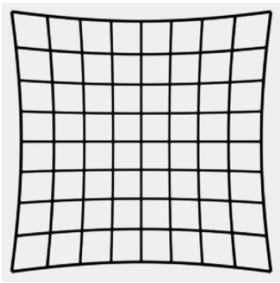
Radial lens distortion

We distort points by a polynomial in the distance from the principal point (the radius ***r***).



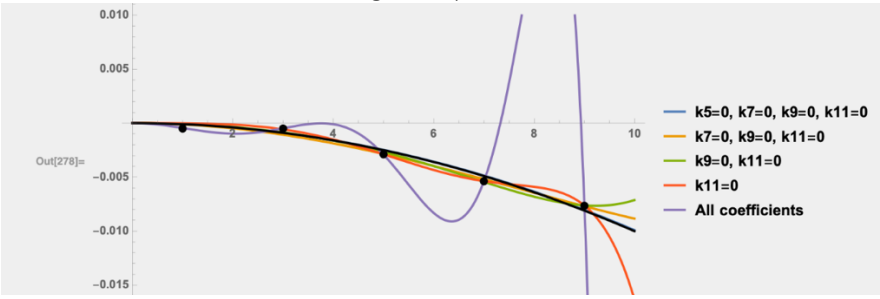
Radial distortion equations:

$$r_d = r(1 + \Delta r(r))$$
$$\Delta r(r) = k_3\|r\|^2 + k_5\|r\|^4 + \cdots$$

Barrel distortional $\Delta r(r) = -0.5\|r\|^2$	Pincushion distortion $\Delta r(r) = +0.5\|r\|^2$
	

Radial lens distortion in practice:

Overfitting /extrapolation errors



$$\Delta r(r) = = -10^{-4}\|r\|^2 + \delta_{error}$$

Almost all lenses have lens distortion!

Projection under radial distortion:
Projection:

$$q = K \ \sqcap^{-1} \left[dist[\sqcap ([R|t]Q)]\right], \text{ where}$$

$$dist(p) = p \cdot (1 + \ \Delta r(\|p\|)$$

$\sqcap \rightarrow$ from homogeneous to inhomogeneous coordinates (from 3D to 2D).
$p \rightarrow$ 2D point in inhomogeneous coordinates.
This follows the math in OpenCV unlike the Lecture Notes.

Undistroting an image:

- Given a distorted image and the distortion coefficients, how do you undistort it?
  - Undistorting the image, is equivalent to finding the intensity in evert undistorted (integer) pixel location p_{ij}.
  - Our distortion function tells us where to sample p_{ij} in the distorted image ($p_{a,ij}$). The intensity can be computed using bilinear interpolation.

- Does it involve inverting the distortion function?
  - Not necessary to invert the distortion function.

Homographies

Consider the following plane:

$$P = aA + bB + C = \begin{bmatrix} A & B & C \end{bmatrix} \begin{bmatrix} a \\ b \\ 1 \end{bmatrix}$$

C is a point on the plane, and ***A*** and ***B*** are vectors that span the plane.

Projections of points on a plane:

$$q = P \, P_h = P \begin{bmatrix} A & B & C \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ 1 \end{bmatrix} = H \begin{bmatrix} a \\ b \\ 1 \end{bmatrix}$$

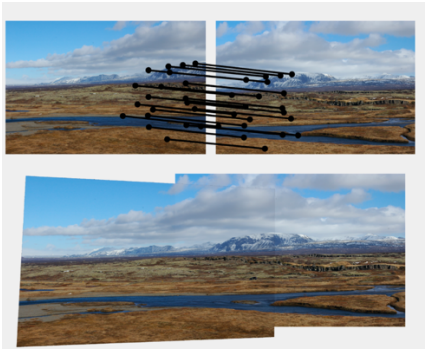
H is a 3 x 3 matrix called a ***homography.***

Homographies between images:
Two cameras photographing the same plane let's us establish pixel correspondence:

$$q_1 = H_1 Q \, , \qquad q_2 = H_2 Q$$
$$q_1 = H_1 H_2^{-1} q_2 = H q_2$$

Two view geometry without baseline:
If two cameras are at the same point we can use a homography to relate their pixels.

Application: image stitching (panorama)



Homography multiplication:
If you multiply a homography *H* by a scalar, how does that change the mapping it describes?
Nothing happens. It doesn't!

$$q_2 = Hq_1$$
$$s\,q_2 = (sH)\,q_1$$

q_2 *and* $s\,q_2$ are the same point.

Homography degrees of freedom:

- 3x3 matrix has 9 numbers
- Scale is arbitrary
  - A homography has 8 degrees of freedom.
- We need 4 pairs of points to estimate a homography
  - Each point pair imposes two constraints (x and y).

Any two images of the same plane are related by a homography.

Homography estimation

Consider a number of points q_{1i} and q_{2i}. The homography relates them as follows:

$$q_{1i} = Hq_{2i}$$

The cross product of two parallel vectors in the zero vector:

$$q_{1i} = Hq_{2i}$$
$$0 = q_{1i}\,x\,Hq_{2i}$$

We can isolate ***H*** in this expression.

$0 = q_{1i}\,x\,Hq_{2i}$ → $B^{(i)}$ flatten (H^T) ,

$$B^{(i)} = \begin{bmatrix} 0 & -x_{2i} & x_{2i}y_{1i} & 0 & -y_{2i} & y_{2i}y_{1i} & 0 & -1 & y_{1i} \\ x_{2i} & 0 & -x_{2i}x_{1i} & y_{2i} & 0 & -y_{2i}x_{1i} & 1 & 0 & -x_{1i} \\ -x_{2i}y_{1i} & x_{2i}x_{1i} & 0 & -y_{2i}y_{1i} & y_{2i}x_{1i} & 0 & -y_{1i} & x_{1i} & 0 \end{bmatrix}$$

flatten $(H^T) = [H_{11} \quad H_{21} \quad H_{31} \quad H_{12} \quad H_{22} \quad H_{32} \quad H_{13} \quad H_{23} \quad H_{33}]^T$

Defining $B^{(i)}$:
The $B^{(i)}$ matrices can be implemented with the Kronecker product:

$$B^{(i)} = q_{2i}^T \otimes [q_{1i}]_x = q_{2i}^T \otimes \begin{bmatrix} 0 & -1 & y_{1i} \\ 1 & 0 & -x_{1i} \\ -y_{1i} & x_{1i} & 0 \end{bmatrix}$$

Homography matrix solution:
Given n corresponding points, we define $\boldsymbol{B}$

$$B = \begin{bmatrix} B^{(1)} \\ B^{(2)} \\ \dots \\ B^{(n)} \end{bmatrix}$$

$$0 = B\ flatten\ (H^T)$$

How do we solve these linear systems of equations?

Homogeneous least-squares solution

- Problem of type $Ax = 0$
- When noise is present, cannot be satisfied exactly
- Equivalent to $\min_x \|Ax\|_2^2$
- Trivial solution x=0 (not useful)
- x can be arbitrarily scaled
    - We impose $\|x\|_2 = 1$

$$\|Ax\|_2^2 = x^T A^T A\ x$$

Assume x is an eigenvector of the square matrix $A^T A$.

Which eigenvector? Eigen-decomposition of $A^T A$:

$$A^T A = V \wedge V^T$$
$$v_i^T A^T A v_i = v_i^T\ V \wedge V^T v_i = \lambda_i$$

So, the minimal solution is the eigenvector with the smallest eigenvalue.
Any other vector will give a larger error.

Least-squares solution with SVD:
Using the SVD (*np.linalg.svd)*

$$A^T A = (U \sum V^T)^T \quad U \sum V^T = V \sum^T U^T U \sum V^T = V \sum^T \sum V^T = V \wedge V^T$$

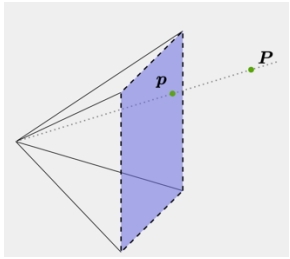
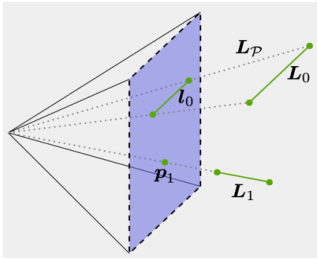
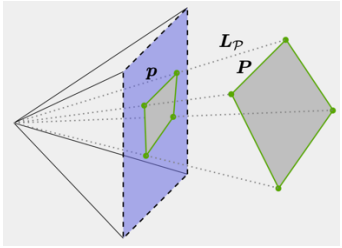
V are the eigenvectors of $A^T A$ and the minimum solution has the *smallest singular value.*

The solution of $Ax = 0$ where $\|x\|^2 = 1$ is found using SVD.
$x = v$ where the singular vector v corresponds to the smallest singular value.

Normalization:
For numerical stability, it is recommended to give both sets of points mean 0 and standard deviation 1 before estimating the homography.

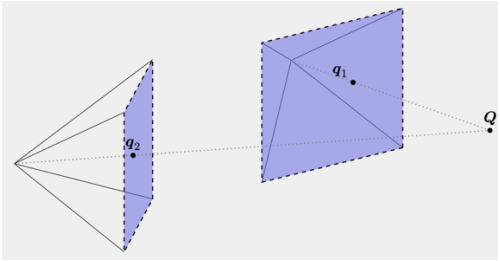
Week 3: Stereo view geometry. Epipolar geometry, triangulation.

Projections of points, lines, and planes

Point projections: Any 3D point P projects to a 2D point p.	
Line projections: Any 3D line L projects to a 2D point l,except if $L \| \| L_{\mathcal{P}}$. $L_0 \xrightarrow{\text{projection}} l_0$, $L_1 \xrightarrow{\text{projection}} p_1$.	
Plane projections: Any 3D plane P projects to the image plane, except if $P \| \| L_{\mathcal{P}}$. *Where then?*	

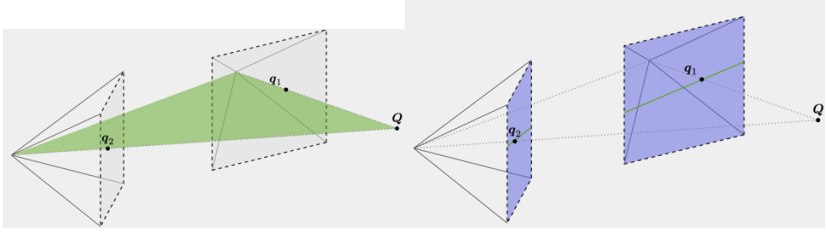
Epipolar planes and lines

Stereo view:



Epipolar planes:

The camera centers and the 3D point Q form an epipolar plane.



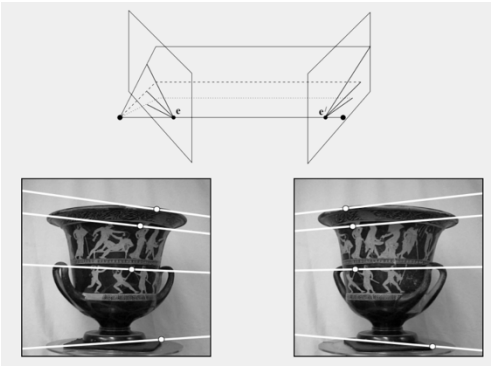
Which line in space is always part of the epipolar planes?

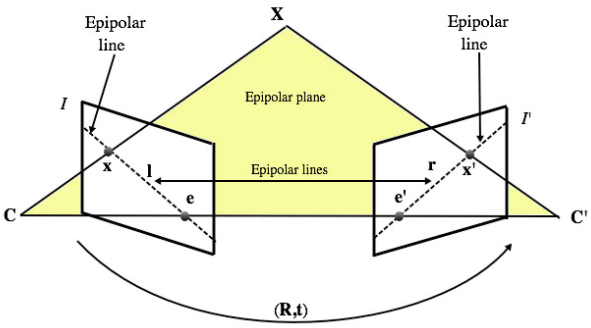
The epipolar lines intersect the epipolar plane and images.

The relation between the line $q_i Q$ and the epipolar lines? All epipolar lines intersect in the epipoles.

Epipoles

Where are the epipoles?





Stereo view is an epipolar geometry:

→ Each Q in 3D has an epipolar plane. Pixels correspond if and only if they lie in the same epipolar plane.

Essential and fundamental matrices

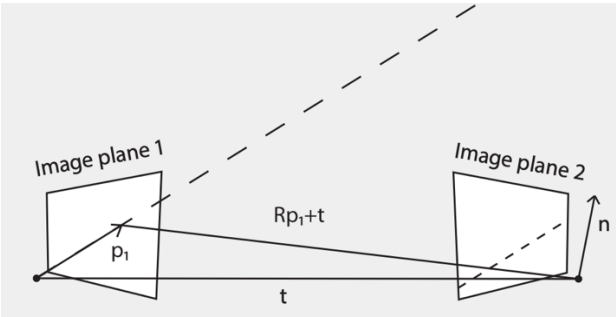
Setting the scene:

Let q refer to a 2D point in pixels and p refer to the same point in 3D in reference frame of the camera. These are related as follows:

$$q = Kp$$
$$p = K^{-1}q$$

The essential matrix:

Consider the epipolar plane given by p_1:



Relative to camera 2, what is the normal of the epipolar plane?

The normal is orthogonal (perpendicular) to t *and* $Rp_1 + t$.

The normal is then:

$$n = tx(Rp_1 + t) = tx(Rp_1) = [t]_x Rp_1 = Ep_1$$

E is called the essential matrix.

- Dot product of orthogonal vectors are zero.
- For the corresponding p_2 in the second camera.

$$0 = p_2^T n = p_2^T E p_1$$

- The essential matrix imposes a constraint on corresponding points.

How to interpret this?

$$p = K^{-1}q$$

Are p_1 and p_2 in homogeneous or inhomogeneous coordinates? There are two interpretations:

1. p_1 and p_2 are 3D points and n is a vector in 3D. We use this in our derivations.
2. p_1 and p_2 are 2D points and $n = E\,p_1$ is the epipolar line, both are in homogeneous coordinates.

The fundamental matrix:

Recall: $p = K^{-1}q$

Then:

$$p_2^T E p_1 = 0$$
$$(K_2^{-1}q_2)^T E (K_1^{-1}q_1) = 0$$
$$q_2^T K_2^{-T} E K_1^{-1} q_1 = 0$$
$$q_2^T F q_1 = 0$$

where $\boldsymbol{F}$ is the fundamental matrix.

!!! The essential and fundamental matrices form requirements for pixel correspondence.

$$p_2^T E p_1 = 0$$
$$q_2^T F q_1 = 0$$

The fundamental matrix contains all the information of the camera matrix. With the fundamental matrix, we can retrieve the intrinsic properties of the camera.

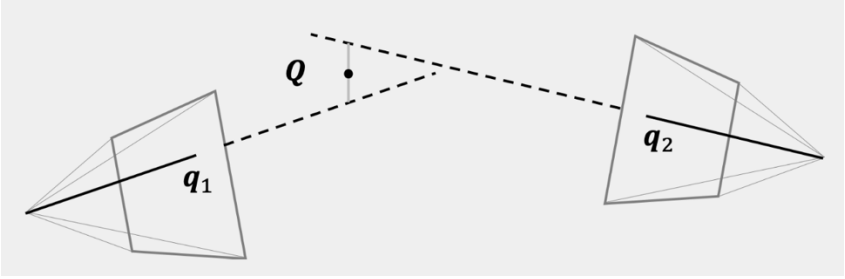
Fundamental/essential matrix vs homography:

- The fundamental and essential matrices yield epipolar lines:
  - $0 = p_2^T E p_1$
  - $0 = q_2^T F q_1$
  - Corresponding points must lie on the epipolar line, no matter what is the image.
- The homography establishes one-to-one correspondences:
  - $q_2 = H q_1$
  - Only valid for planes.

Degrees of freedom of F:

- F has 9 numbers, and is scale invariant.
- $[t]_x$ has rank 2 ,and thus F has as well. In other terms: $det(F) = 0$.

Triangulation



We've seen the same point in many (known) cameras and want to find the point in 3D.

Because of noise, there is not always a solution without error.

Triangulation problem:

Consider a projection matrix:

$$P_i = \begin{bmatrix} p_i^{(1)} \\ p_i^{(2)} \\ p_i^{(3)} \end{bmatrix}$$

Triangulation equations:

Projection gives the pixels

$$q_i = \begin{bmatrix} s_i x_i \\ s_i y_i \\ s_i \end{bmatrix} = P_i Q = \begin{bmatrix} p_i^{(1)} Q \\ p_i^{(2)} Q \\ p_i^{(3)} Q \end{bmatrix}$$

This is two constrains (x_i, y_i)in three equations.

As $s = p_i^{(3)} Q$, we have

$$(p_i^{(3)} Q) \begin{bmatrix} x_i \\ y_i \end{bmatrix} = P_i Q = \begin{bmatrix} p_i^{(1)} Q \\ p_i^{(2)} Q \end{bmatrix}$$

Rearranged into

$$0 = \begin{bmatrix} p_i^{(3)} x_i - p_i^{(1)} \\ p_i^{(3)} y_i - p_i^{(2)} \end{bmatrix} Q = B^{(i)} Q$$

Triangulation solutions:

Define B:

$$B = \begin{bmatrix} B^{(1)} \\ B^{(2)} \\ \dots \\ B^{(n)} \end{bmatrix} = \begin{bmatrix} p_1^{(3)} x_1 - p_1^{(1)} \\ p_1^{(3)} y_1 - p_1^{(2)} \\ p_2^{(3)} x_2 - p_2^{(1)} \\ p_2^{(3)} y_2 - p_2^{(2)} \\ \dots \end{bmatrix}$$

Use SVD to find $\underset{Q}{\operatorname{argmin}} \|BQ\|_2$, s.t. $\|Q\|_2 = 1$

Linear algorithms:

- $\underset{Q}{\operatorname{argmin}} \|BQ\|_2$, s.t. $\|Q\|_2 = 1$
- This is a linear algorithm used to solve the problem.
- Which error would we actually like to minimize?
  - Depends why we believe that there are errors
  - Errors in the observed pixel location

Triangulation: ideal error

- Let $[\tilde{x}_i \quad \tilde{y}_i]$ and $[x_i \quad y_i]$ refer to the observed and projected pixel coordinates, respectively.
- Which error would we like to minimize?

$$e_{ideal} = \sum_i \left\| \begin{bmatrix} \tilde{x}_i \\ \tilde{y}_i \end{bmatrix} - \begin{bmatrix} x_i \\ y_i \end{bmatrix} \right\|_2^2$$

$$e_{algebratic} = \sum_i \|B_i Q\|_2^2 = \sum_i \left\| \begin{bmatrix} p_i^{(3)} \tilde{x}_i - p_i^{(1)} \\ p_i^{(3)} \tilde{y}_i - p_i^{(2)} \end{bmatrix} Q \right\|_2^2 =$$

$$= \sum_i \left\| p_i^{(3)} Q \begin{bmatrix} \tilde{x}_i \\ \tilde{y}_i \end{bmatrix} - \begin{bmatrix} p_i^{(1)} \\ p_i^{(2)} \end{bmatrix} Q \right\|_2^2 =$$

$$= \sum_i \left\| s_i \begin{bmatrix} \tilde{x}_i \\ \tilde{y}_i \end{bmatrix} - s_i \begin{bmatrix} x_i \\ y_i \end{bmatrix} \right\|_2^2$$

Error terms compared:

- We can rewrite the terms slightly again

$$e_{ideal} = \sum_i (\hat{x}_i - x_i)^2 + (\hat{y}_i - y)^2$$

$$e_{algebraic} = \sum_i s_i{}^2(\hat{x}_i - x_i)^2 + s_i{}^2(\hat{y}_i - y)^2$$

- s_i is larger for cameras that are further from Q.

Linear vs non-linear algorithms:

- e_{ideal} can be minimized using non-linear optimization
- Linear algorithms are very fast. Only minimizes an algebraic error.
- We can estimate many things using linear algorithms.
    - Triangulation, homography, fundamental matrix, projection matrix
    - They all have the problem that they don't minimize the exact error we desire.
- Linear algorithms are acceptable in most cases.
- When high accuracy is desired initialize the non-linear optimization with the linear solution.

Week 4: Camera calibration

Direct linear transformations

Start from the projection equation

$$q_i = PQ_i$$
$$\begin{bmatrix} sx_i \\ sy_i \\ s \end{bmatrix} = P \begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix}$$

Then rearrange into the form $B^{(i)}\ \text{flatten}\ (P^T) = 0$.

$$\begin{aligned} q_i &= PQ_i, \\ 0 &= q_i\ x\ PQ_i, \\ &= [q_i]_x\ \ PQ_i, \\ &= B^{(i)}\ \text{flatten}\ (P^T), \end{aligned}$$

$$B^{(i)} = \begin{bmatrix} 0 & -X_i & X_i y_i & 0 & -Y_i & Y_i y_i & 0 & -Z_i & Z_i y_i & 0 & -1 & y_i \\ X_i & 0 & -X_i x_i & Y_i & 0 & -Y_i x_i & Z_i & 0 & -Z_i x_i & 1 & 0 & -x_i \\ -X_i y_i & X_i x_i & 0 & -Y_i y_i & Y_i x_i & 0 & -Z_i y_i & Z_i x_i & 0 & -y_i & x_i & 0 \end{bmatrix},$$

$$= Q_i \otimes [\ q_i\ ]_x$$

$$flatten\ (P^T) = [\mathcal{P}_{11} \quad \mathcal{P}_{21} \quad \mathcal{P}_{31} \quad \mathcal{P}_{12} \quad \mathcal{P}_{22} \quad \mathcal{P}_{32} \quad \mathcal{P}_{13} \quad \mathcal{P}_{23} \quad \mathcal{P}_{33} \quad \mathcal{P}_{14} \quad \mathcal{P}_{24} \quad \mathcal{P}_{34}]^T$$

Now let

$$0 = B\ \text{flatten}\ (P^T)$$

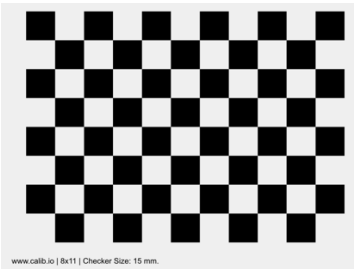
Where

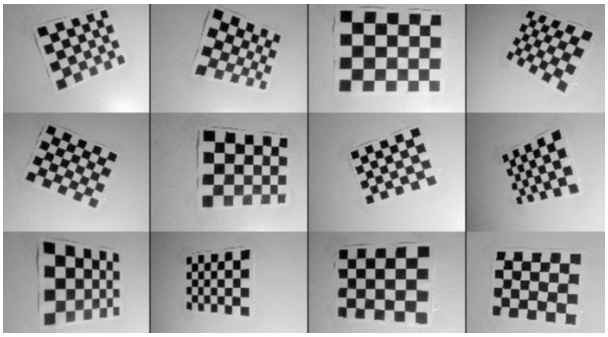
$$B = \begin{bmatrix} B^{(1)} \\ B^{(2)} \\ \dots \end{bmatrix}$$

And solve using SVD on B.

Zhang's method (2000)

Using a checkerboard, view the checkerboard in different poses:





Problem: Each view i has a different rotation R_i and translation t_i.
→ How do we find all P_i ?

Zhang's Solution:

1. First, assume all checkboard corners are in the $Z = 0$ plane:

$$Q_j = \begin{bmatrix} X_j \\ Y_j \\ 0 \end{bmatrix}$$

2. Simplifying the projection equation.

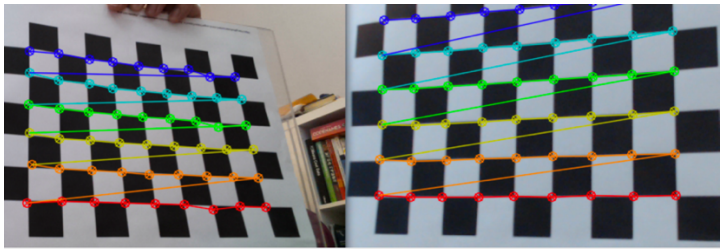
- Let $r_i^{(c)}$ is the c[th] column of R_i. Now projection is:

$$q_{ij} = P_i Q_j = K \begin{bmatrix} r_i^{(1)} & r_i^{(2)} & r_i^{(3)} & t_i \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ 0 \\ 1 \end{bmatrix} = K \begin{bmatrix} r_i^{(1)} & r_i^{(2)} & t_i \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ 1 \end{bmatrix}$$

3. From projections to homographies.

$$q_{ij} = K \begin{bmatrix} r_i^{(1)} & r_i^{(2)} & t_i \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ 1 \end{bmatrix} = H_i \tilde{Q}_j$$

- The homographies H_i can bet determined from the plane-plane correspondence $\tilde{Q}_j$ to q_{ij}. (week 2).
- Corner correspondences: Need to find unique positions for corners q_{ij}.



→ Find all H_i from corners $\tilde{Q}_j$ and projections q_{ij} with $q_{ij} = H_i \tilde{Q}_j$. For example, using SVD.

From homographies to the camera matrix:

$$H_i = \begin{bmatrix} h_i^{(1)} & h_i^{(2)} & h_i^{(3)} \end{bmatrix} = \lambda K \begin{bmatrix} r_i^{(1)} & r_i^{(2)} & t_i \end{bmatrix}$$

$r_i^{(1)}$ and $r_i^{(2)}$ are orthonormal, i.e.

$$r_i^{(1)T} r_i^{(1)} = r_i^{(2)T} r_i^{(2)} = 1,$$
$$r_i^{(1)T} r_i^{(2)} = r_i^{(2)T} r_i^{(1)} = 0.$$

Express $r_i^{(\alpha)}$ using $h_i^{(\alpha)}$:

$$h_i^{(\alpha)} = \lambda K r_i^{(\alpha)} \Leftrightarrow K^{-1} h_i^{(\alpha)} = \lambda r_i^{(\alpha)}$$

Now the constrains from the equations below are:

$$h_i^{(1)T} K^{-T} K^{-1} h_i^{(2)} = 0$$
$$h_i^{(1)T} K^{-T} K^{-1} h_i^{(1)} = h_i^{(2)T} K^{-T} K^{-1} h_i^{(2)} = \lambda^2$$

We have found constrains on the camera matrix.

Number of constraints:

- Two constraints don't seem that impressive.
- Homography has eight degrees of freedom.
- Pose of checkboard has six (3 rotation, 3 translation)
- One homography can only fix two degrees of freedom of the camera matrix.

Define some new variables i:

How to put into practice?

Define the matrix:

$$B = K^{-T}K^{-1} = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{12} & B_{22} & B_{23} \\ B_{13} & B_{23} & B_{33} \end{bmatrix},$$

$$b = [B_{11} \quad B_{12} \quad B_{22} \quad B_{13} \quad B_{23} \quad B_{33}]^T$$

Now define $v_i^{(\alpha\beta)}$ such that

$$v_i^{(\alpha\beta)}\, b = h_i^{(\alpha)^T} B\, h_i^{(\beta)}$$

Then $v_i^{(\alpha\beta)}$ must be an 1 x 6 vector given by

$$v_i^{(\alpha\beta)} = \left[h_i^{(1\alpha)}h_i^{(1\beta)}, \quad h_i^{(1\alpha)}h_i^{(2\beta)} + h_i^{(2\alpha)}h_i^{(1\beta)}, \quad h_i^{(2\alpha)}h_i^{(2\beta)}, \quad \dots \quad h_i^{(3\alpha)}h_i^{(1\beta)} + h_i^{(1\alpha)}h_i^{(3\beta)}, \quad h_i^{(3\alpha)}h_i^{(2\beta)} + h_i^{(2\alpha)}h_i^{(3\beta)} \quad h_i^{(3\alpha)}h_i^{(3\beta)}\right]$$

where $h_i^{(rc)}$ is the element in row r and column c of H_i.

Recall our constraints:

$$h_i^{(1)^T} K^{-T}K^{-1} h_i^{(1)} = h_i^{(2)^T} K^{-T}K^{-1} h_i^{(2)}$$

$$h_i^{(1)^T} B\, h_i^{(1)} - h_i^{(2)^T} B\, h_i^{(2)} = 0$$

$$(v_i^{(11)} - v_i^{(22)})\, b = 0$$

Now we can express our constraints in matrix-form:

$$\begin{bmatrix} v_i^{(12)} \\ v_i^{(11)} - v_i^{(22)} \end{bmatrix} b = 0$$

For all checkerboard poses:

$$V\, b = \begin{bmatrix} v_1^{(12)} \\ v_1^{(11)} - v_1^{(22)} \\ v_2^{(12)} \\ v_2^{(11)} - v_2^{(22)} \\ \dots \end{bmatrix} b = 0$$

When b is found then K can be determined using the formulas in Zhang's paper.

→ Find the camera matrix K through b using $Vb = 0$ and SVD, where V is built from the homographies H_i

From the homographies to poses:

Recall:

$$H_i = \begin{bmatrix} h_i^{(1)} & h_i^{(2)} & h_i^{(3)} \end{bmatrix} = \lambda K \begin{bmatrix} r_i^{(1)} & r_i^{(2)} & t_i \end{bmatrix}$$

Now we can recover R_i *and* t_i:

$$r_i^{(1)} = \frac{1}{\lambda} K^{-1} h_i^{(1)},$$

$$r_i^{(2)} = \frac{1}{\lambda} K^{-1} h_i^{(2)},$$

$$t_i = \frac{1}{\lambda} K^{-1} h_i^{(3)},$$

$$r_i^{(3)} = r_i^{(1)}\, x\, r_i^{(3)}$$

Where $\lambda = \left\| K^{-1} h_i^{(1)} \right\|_2 = \left\| K^{-1} h_i^{(2)} \right\|_2$

$$t_i = \frac{1}{\lambda} K^{-1} h_i^{(3)}$$

What happens if $t_{iz} < 0$? And what does that mean?

Can we ensure correctness?

(Run the code again with $-H_i$ (flipped sign).

Non-linear calibration

Least-squares method:
With projection equation,

$$q_{ij} = K\,[R_i \quad t_i]Q_j$$

Then: $E = \sum_{i,j} \left\| dist\,(\pi\,(\,K\,[R_i \quad t_i]Q_j\,)\,) - \pi\,(\,q_{ij})\right\|^2$

Solution: minimize E w.r.t K , R_i , t_i and lens distortion.

Practical remarks

- What images should we take?
  - Our two constraints per image are based on $r_i^{(1)}$ and $r_i^{(2)}$.
  - Two parallel checkboards express the same constraints.
    (at least when we don't consider lens distortion)
  - Make sure to rotate the checkboards so they are not parallel.
  - Make the checkboards take up as much of the frame as possible.

- How many images?
  - Without lens distortion: In theory at least three.
  - In practice: It depends on…
    - Some people use 2000+ images for single calibration.
    - Look at the projection error

Week 5: Nonlinear optimization and camera calibration

Non-linear least-squares optimization

Problem of the form

$$\min_x \|g(x) - y\|_2^2$$

Make all parameters into a vector x and optimize everything.

- Homography estimation
- Pose estimation
- Bundle adjustment
- Camera calibration with lens distortion
- Triangulation (week 5)

Reformulate a bit: *(To obtain single squares-numbers)*

$$e(x) = \|g(x) - y\|_2^2 = \|f(x)\|_2^2 = f(x)^T f(x)$$

Levenberg-Marquardt:

We solve the problem in an iterative fashion. At the kith k^{th} iteration:
→ Replace with f with first order approximation around x_k.

$$f(x_k + \delta) = f(x_k) + \boldsymbol{J}\,\delta$$

J is the Jacobian that contains all first order derivatives of f at x_k.
Note that the Jacobian of f and g are identical.

The sum of squared errors is then:

$$e(x_k + \delta) = \|f(x_k + \delta)\|_2^2 = (f(x_k) + J\delta)^T (f(x_k) + J\delta) =$$
$$= f(x_k)^T f(x_k) + 2(J\delta)^T f(x_k) + (J\delta)^T J\delta^T =$$
$$= f(x_k)^T f(x_k) + 2\delta^T J^T f(x_k) + \delta^T J^T J\delta^T$$

which is a second order approximation of e using only first order derivatives of f.

1. Take the derivative of $e(x_k + \delta)$:
$$\frac{\partial e((x_k + \delta)}{\partial \delta} = 2J^T f(x_k) + 2J^T J\delta$$
2. Finding the optimum by setting the derivative equal to zero
$$0 = 2J^T f(x_k) + 2J^T J\delta$$
$$J^T J\delta = -J^T f(x_k)$$
3. Solve for δ and then set $x_{k+1} = x_k + \delta$.
4. Rinse and repeat.

The λ parameter:
The second order approximation of e is better the closer we are to the minimum. Otherwise, it is not properly working, reason why we add the parameter λ at the diagonal of the matrix.

When far away, it can be quite bad, in which case we want to take a small step in the gradient direction.

$$(J^T J + \lambda I)\delta = -J^T f(x_k)$$

Decrease λ when $f(x_{k+1}) < f(x_k)$ and increase when not.

Summary: Levenberg-Marquardt is not the only algorithm for nonlinear optimization, but it is often well suited to the types of problems we are likely to encounter in computer vision.
The nature of the least-squares problem is exploited to get a second order method, using only first order derivatives.

Gradients

How do we compute J ? (The Jacobian values of f)
It is just a lot of derivatives!

- Analytical gradients
- Automatic differentiation
  - Reverse mode automatic differentiation (backpropagation)
  - Forward mode automatic differentiation (dual numbers)
- Finite differences approximation

Analytical gradients

Get out pen and paper (or Maple) and find the exact derivative.

The reprojection error of a point in 3D in multiple cameras

$$f(Q) = \begin{bmatrix} \Pi(P_1\Pi^{-1}(Q)) - \tilde{q}_1 \\ \dots \\ \Pi(P_n\Pi^{-1}(Q)) - \tilde{q}_n \end{bmatrix}$$

- f returns a vector of length $2n$.
- Q has three parameters
- Thus J is $2n \times 3$

Triangulation example:
The projected point in homogeneous coordinates:

$$\mathrm{q} = P\,\Pi^{-1}(Q) = PQ_h = \begin{bmatrix} \mathrm{sx} \\ sy \\ \mathrm{s} \end{bmatrix} = \begin{bmatrix} \mathrm{p}^{(1)} \\ \mathrm{p}^{(2)} \\ \mathrm{p}^{(3)} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Let's focus on x:

$$x = \frac{p^{(1)}Q_h}{p^{(3)}Q_h} = \frac{P_{11}X + P_{12}Y + P_{13}Z + P_{14}}{P_{31}X + P_{32}Y + P_{33}Z + P_{34}}$$

A single element of J is then given by:

$$\frac{d}{dX}x = \frac{P_{11}}{p^{(3)}Q_h} - \frac{P_{31}(p^{(1)}Q_h)}{(p^{(3)}Q_h)^2}$$

Summary:
Analytical gradients are extremely useful
+ very fast
+ accurate
- complicated to derive and implement
Many functions in OpenCV return the Jacobian in addition to the values themselves.

Finite differences approximation

Forward differences:
A first order Taylor expansion of f

$$f(x+h) = f(x) + \frac{d}{dx}f(x)\,h + O(h)$$

Can be rewritten to

$$\frac{d}{dx}f(x) = \frac{f(x+h) - f(x)}{h} + O(h)$$

This is called forward differences. (2-point)

Central differences:

Using a second order Taylor expansion of f evaluated at $x - h$ and $x + h$ can be rearranged to yield

$$\frac{d}{dx}f(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2)$$

This is called central differences, which is more accurate $(O(h^2))$, but requires two new evaluations of f.

In practice:

- Principle can only be applied to one parameter at a time.
- To compute J you need to evaluate f once (or twice) for each element in J (which is not a lot)
- Number of evaluations can be reduced if you know the sparsity structure of J, i.e., which elements of x that affect which elements of $f(x)$.

Summary:
They only give an approximation of gradients
+ convenient
- slow
- has parameter h
- numeric problems
Only use when speed and robustness are not your primary concerns.

Rotations in optimization

- Rotations are usually 3x3 matrices but have just three degrees of freedom.
- How can we parametrize rotations?
    - Optimizing all 9 numbers gives something that is no longer a rotation matrix.

Euler angles:

Euler angles $(\theta_x, \theta_y, \theta_z)$uses only three numbers

- Suffers from gimbal lock, i.e., that one parameter does nothing in certain configurations.
    - Unsuitable for optimization
- If the rotation is close to identity, we are fare from gimbal lock
    - No problems in this case
    - Optimize over a rotation relative to the initial guess
    - $\theta_x, \theta_y, \theta_z$ will usually stay close to zero
        - Only if initial guess is good
        - For finite difference start from $\theta_x = \theta_y = \theta_z = 2\pi$

Axis-angles:

- Axis-angle represents a rotation as a rotation of θ around an axis v, where $\|v\|_2 = 1$
- This is then stored as a vector v/θ (only three elements)
- This is how OpenCV return most rotations (`rvec`), and it can be converted to a full rotation matrix with `cv2.Rodfrigues.`
- Still has weird behavior when interpolating between rotations but works in most cases.

Quaternions:

- Quaternions are the gold standard for representing rotations.
- A quaternion uses four numbers representing a rotation (a, i, j, k) subject to $a^2 + i^2 + j^2 + k^2 = 1$
- Smooth and suffers from no problems, however we have to normalize the quaternion each optimization step, to ensure it is still a valid quaternion.

Software packages:

There are many implementations out there.

- Ceres (C++)
  - Dual numbers (no implementing derivatives)
  - Quaternion parameterization
- `spiciy.optimize.least_squares`
  - Easy to use
  - Finite differences or analytical gradients

Camera calibration

More tips and tricks:

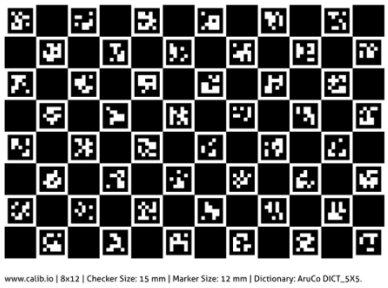
- Checkboard alternatives
- Subpixel estimation
- Overfitting / cross-validation
- Bundle adjustment

Checkboard

OpenCV needs to see all corners of the checkerboard in order to detect it.

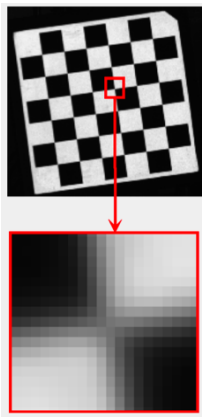
Getting the entire image plane converted in detected points is hard, especially near the edges.

A *ChArUco* (2D data matrix) boards is a checkboard with ArUco markers.



Possible to detect partial checkboards!

Subpixel corner estimation



- Where is the corner?
- Look at a neighborhood around a corner
- OpenCV has `cv2.cornerSubPix` but it is not good.

What to do?

Good method for subpixel corner estimation:
A good subpixel corner estimator is presented in Schops et al.[1]

1Schops, Thomas, et al. "Why having 10,000 parameters in your camera model is better than twelve." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020.

Generate n random 2D vectors relative to the corner s_i.

$$C_{sym}(H) = \sum_{i=1}^{n}((I(H(s_i)) - I(H(-s_i)))^2)$$

H maps a point using the homography, and I interpolates the value of the image at a given point.
Minimize C_{sym}.

A cautionary tale:

- Including more distortion parameters in your model will always give a lower re-projection error.
- For the images used calibration! (will not necessarily generalize)
- This can be overfitting.
- Use cross-validation.

Cross validation

How to do cross-validation:

- For each checkerboard, split your detected corners into a validation and training set.
- Do the calibration using all training corners.
- Use the estimated checkerboard pose to reproject the validation errors.

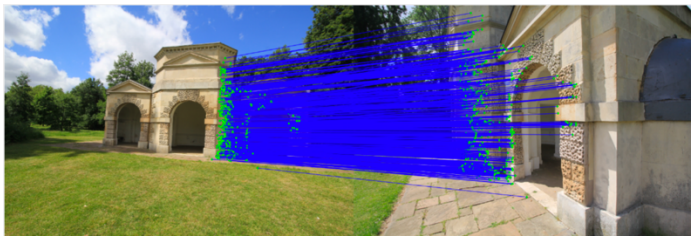
Bundle adjustment

- The checkerboard can have imperfections, such as not being flat.
    - The checkerboards you get today are not very flat.
- Optimize everything again, including the 3D positions of the corners.

Week 6: Simple Feature

Image correspondence problem

The problem of matching two parts of different images.



More images of the same scene:

- Correspondence exists between parts that are visible in each image.
- Not all corresponding points have unique pattern
- Idea: Only choose points that can be identified uniquely.

What are good points?

- When we move the camera there are also points that change appearance.
- If we can choose some points that have the same appearance in all images, it would be better.
- We can identify the same corners in multiple image: *Corner detector*, detecting corners.

Problems:

1. Movement of camera:
    - Scale: distance of the camera to object
    - Rotation: objects and camera are rotated between frames
    - Translation: movement of camera from one place to another

Result: Appearance changes dependent on distance to camera

2. Other issues:
    - Occlusion: objects can be in front of other objects
    - Lighting change: darker/lighter, color of light (change in spectrum), direction of light
    - Clutter: many objects in a scene

Solutions:

Invariance to imaging problems.

- Focus on points and a small area abound each point.
- Two aspects:
    - 1. Identify key points
    - 2. Characterize pattern around point

Features and feature descriptors:

First some terminology:

- Key points, interest points, and feature points are the anem – namely points in an image with a coordinate position (also between pixels).
- Descriptors, key point descriptors, interest point descriptors, and feature descriptors are the same and used for characterizing the pattern around a point. Usually, a vector that can be matched between images.

Image patch feature descriptor

One easy wat of describing a feature is to use the local pixel information around the feature.

The image patch matching

Stretch the image patch into a vector f, and this is your descriptor.

A simple comparison operator $d(f_1, f_2)$ just uses the scalar product as the distance between features:

$$d(f_1, f_2) = f_1^T f_2$$

Good with only small transformation, low noise, and unchanged environment.

Cross correlation

A better comparison $X(f_1, f_2)$ uses the cross correlation as the distance:

$$X(f_1, f_2) = \frac{cov\,(f_1, f_2)}{\sqrt{var(f_1) + var(f_2)}}$$

Where

$$cov\,(p, q) = \frac{1}{N-1}\sum_{i}^{N}(p_i - \bar{p})^T(q_i - \bar{q})$$

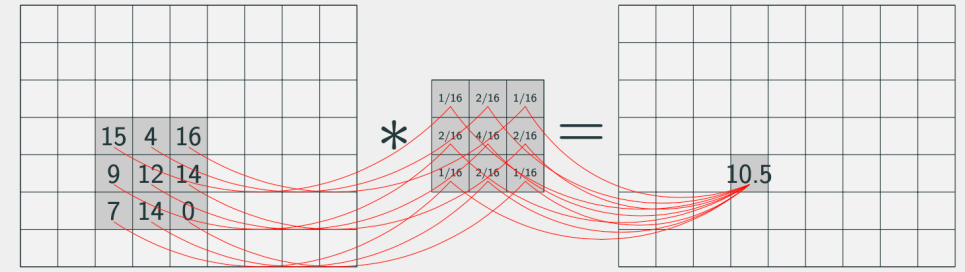
$$var\,(p) = \frac{1}{N-1}\sum_{i}^{N}\|p_i - \bar{p}\|^2$$

- The cross correlation picks up local variation and is therefore more robust against environment changes than squared distance.
- Only useful when images are extremely similar.

Simple features

Filtering and convolution

$$I(x;t) = \int_{\xi\in\mathbb{R}^2} I(x-\xi)g(\xi;t)d\xi \approx \sum_{i=-k}^{k}\sum_{j=-k}^{k} I(x-i, y-j)g(i,j)$$



2D convolution:

We use filtering and convolution for the same operation and use the symbol * for convolution. Convolution is commutative.

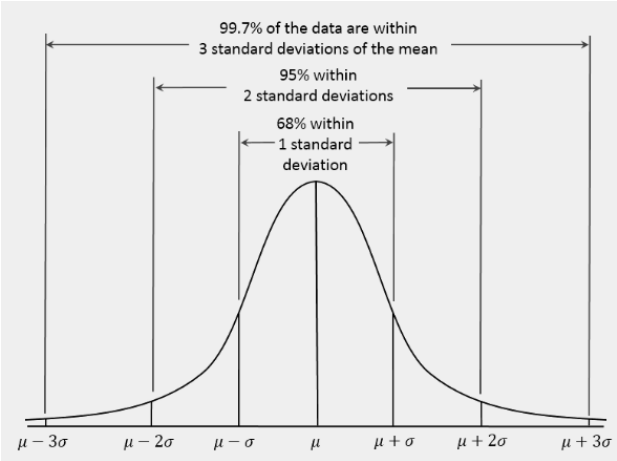
$$I_g = g * I = I * g$$

2D Gaussian $g: \mathbb{R}^2 \; x \; \mathbb{R}_+ \rightarrow \mathbb{R}$

$$g(x, y; \sigma^2) = \frac{1}{2\pi\sigma^2} exp(\frac{-(x^2 + y^2)}{2\sigma^2})$$

- *Separability:*
  Useful property: The (isotropic) Gaussian is separable.
  $$g(x, y; \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} exp(\frac{-x^2}{2\sigma^2}) \frac{1}{\sqrt{2\pi\sigma^2}} exp(\frac{-y^2}{2\sigma^2})$$
  This separability means that
  $$I_g = (g * g^T) * I = g * (g^T * I)$$
  Where g is a vector of the Gaussian.

- *Size of Gaussian filter*
  Ideally: infinitely big – but this is impractical
  Empiric rule: $3\sigma, 4\sigma \; or \; 5\sigma$ rule – example:
  - If $\sigma = 2$, filter size of Gaussian is 2 · 5 · 2+ 1 = 21
  - If $\sigma = 20$, filter size of Gaussian is 2 · 5 · 20 + 1 = 20



Derivatives of an image using Gaussians:

How can we find the derivate of an image in e.g., the x-direction?

Let g be column vector of Gaussian. Consider the blurred image.

$$I_b = g * g^T I$$

Then, the derivative of I_b in the x-direction is

$$\frac{\partial}{\partial x} I_b = \frac{\partial}{\partial x}(g * g^T I) = g * (\frac{\partial}{\partial x} g^T) I = g * g_d^T I$$

Where g_d is the derivative of g. We can compute $\frac{\partial}{\partial y} I_b$ similarly.

Recall the one-dimensional Gaussian:

$$g(x) = \frac{1}{\sqrt{2\pi\sigma^2}} exp(\frac{-x^2}{2\sigma^2})$$

We can compute then the derivative

$$g_d(x) = \frac{d}{dx} g(x) = \frac{-x}{\sigma^2} g(x)$$

Harris corners

Corners and blobs are great features because they are easy to describe and detect.
Harris corners are detected through as points with locally maximum change from a small shift.

$$\Delta I(x,y) = I(x,y) - I(x+\Delta x, y+\Delta y)$$

We define a corner having a large $\Delta I(x,y)^2$, and we average the value over a small area to ensure robustness against noise.
The corner measure is:

$$c(x,y) = g * \Delta I(x,y)^2 = g * (I(x,y) - I(x+\Delta x, y+\Delta y))^2$$

where $g * \dots$ is the convolution with the Gaussian.

Replace with Taylor approximation:

$$I(x+\Delta x, y+\Delta y) \approx I(x,y) - \frac{\partial I}{\partial x}(x,y)\Delta x - \frac{\partial I}{\partial y}(x,y)\Delta y = I(x,y) - I_x \Delta x - I_y \Delta y$$
$$= I(x,y) - \begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} \Delta_x \\ \Delta_y \end{bmatrix}$$

I_x and I_y are also the values at (x,y), but we omit it for readability.

Harris corner measure derivation:
The corner measure is then

$$c(x,y) = g * (I(x,y) - I(x+\Delta x, y+\Delta y))^2 \approx g * \left(\begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} \Delta_x \\ \Delta_y \end{bmatrix} \right)^2$$
$$= g * \left(\begin{bmatrix} \Delta_x & \Delta_y \end{bmatrix} \begin{bmatrix} I_x \\ I_y \end{bmatrix} \begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} \Delta_x \\ \Delta_y \end{bmatrix} \right)$$
$$= g * \left(\begin{bmatrix} \Delta_x & \Delta_y \end{bmatrix} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} \Delta_x \\ \Delta_y \end{bmatrix} \right)$$

Finally, we take the convolution with the Gaussian inside

$$c(x,y) \approx g * \left(\begin{bmatrix} \Delta_x & \Delta_y \end{bmatrix} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} \Delta_x \\ \Delta_y \end{bmatrix} \right) =$$
$$= \left(\begin{bmatrix} \Delta_x & \Delta_y \end{bmatrix} \begin{bmatrix} g*(I_x^2) & g*(I_x I_y) \\ g*(I_x I_y) & g*(I_y^2) \end{bmatrix} \begin{bmatrix} \Delta_x \\ \Delta_y \end{bmatrix} \right)$$
$$= \left(\begin{bmatrix} \Delta_x & \Delta_y \end{bmatrix} C(x,y) \begin{bmatrix} \Delta_x \\ \Delta_y \end{bmatrix} \right)$$

The averaged Hessian:
We end up with an **averaged Hessian matrix $C(x,y)$.**
Corners have a large $c(x,y)$ regardless of the direction of $[\Delta_x \quad \Delta_y]$.

$$c(x,y) = \begin{bmatrix} \Delta_x & \Delta_y \end{bmatrix} \begin{bmatrix} a & c \\ c & b \end{bmatrix} \begin{bmatrix} \Delta_x \\ \Delta_y \end{bmatrix}$$

When is this case?

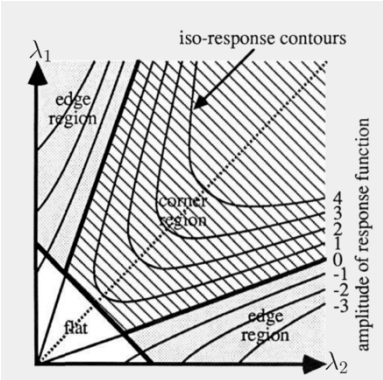
- When both eigenvalues of $C(x,y)$ are large.

The Harris corner metric:
Let λ_1 and λ_2 be the eigenvalues of $C(x,y)$.
We then have the following

$$det(C(x,y)) = \lambda_1 \lambda_2$$
$$trace(C(x,y)) = \lambda_1 + \lambda_2, let\ us\ then\ define$$
$$r(x,y) = det(C(x,y)) - k\ trace(C(x,y))$$
$$r(x,y) = \lambda_1 \lambda_2 - k\ (\lambda_1 + \lambda_2)^2$$

k is a free parameter, typically $k = 0.06$. Notice that $r(x, y)$ is:

- Negative for one eigenvalue much greater than the other
- Large and positive for large eigenvalues
- Small and positive for small eigenvalues



The Harris corner detector:

We can now detect corners by finding points where $r(x, y)$ is greater than some threshold τ.

Typically, you can choose τ to be

$$0.1 \cdot max(r(x, y)) < \tau < 0.8 \cdot max\,(r(x, y))$$

Non-maximum suppression:

Find the local maximum of one pixel compared to neighbors

$$\big(I(x, y) - I(x', y')\big) > 0 \forall x' \in \eta(x, y)$$

Where $\eta(x, y)$ is a neighborhood around the point (x, y).

Suggested procedure:

- Initialize an array of size of image with ones
- Compare entire image with one neighbour (e.g., to the right)
- Set pixels to zero where it is not larger than neighbour
- Find coordinate of pixels with value one.

Canny edges

Often, we might also consider detecting lines.

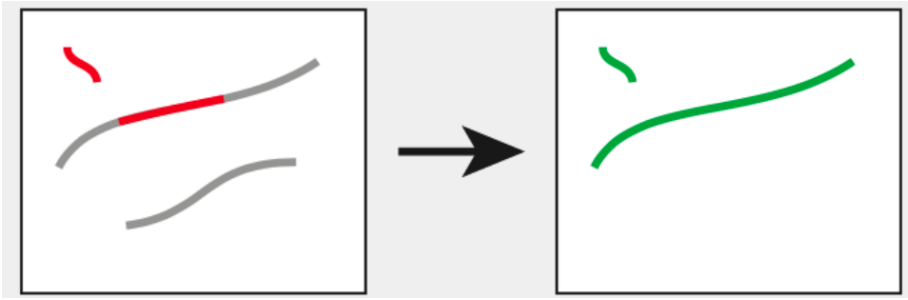
To detect (non-straight) lines, we can use the Canny edge detector.

The metric in the canny edge detector is imply the gradient magnitude

$$m(x, y) = \sqrt{I_x^2(x, y) + I_y^2(x, y)}$$

And the edges are detected by thresholding the magnitude $m(x, y)$, however, in two stages: *seed and grow.*

- Seed: Label edges by a threshold where $m(x, y) > \tau_1$
- Grow: with a second threshold where $m(x, y) > \tau_2$, label if the new points are next to previously labelled edges.
- The threshold values are chosen such that $\tau_1 > \tau_2$



Week 7: Robust Model Fitting

Fitting models:

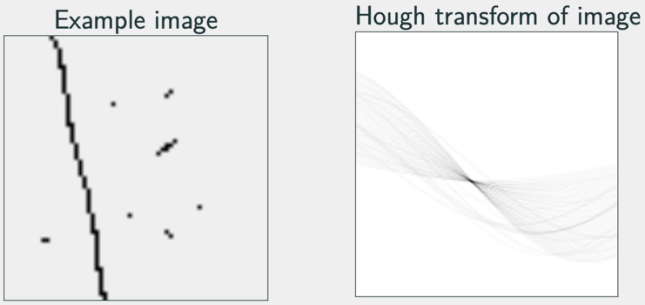
Can we fit a straight line?
Can we fit a straight line when there are a few outliers? Not really… We get a very bad fit. We need robust ways to fit models.

About lines:

- This presentation will use fitting straight lines to data for all examples.
- The same principles generalize to other models.

Hough Transform

It is a transformation of an edge image where lines can be extracted.



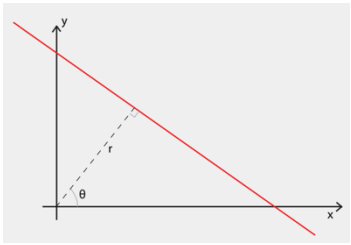
How to represent a line?

$$y = ax + b$$

- Has singularities for vertical lines
- Then homogeneous coordinates? Is over-parametrized.

r, θ representation:
Representation using:

- θ: angle of the line
- r: closest distance from origin to line



Closely related to the homogeneous line representation:

$$l = [cos(\theta) \quad sin(\theta) \quad -r]$$

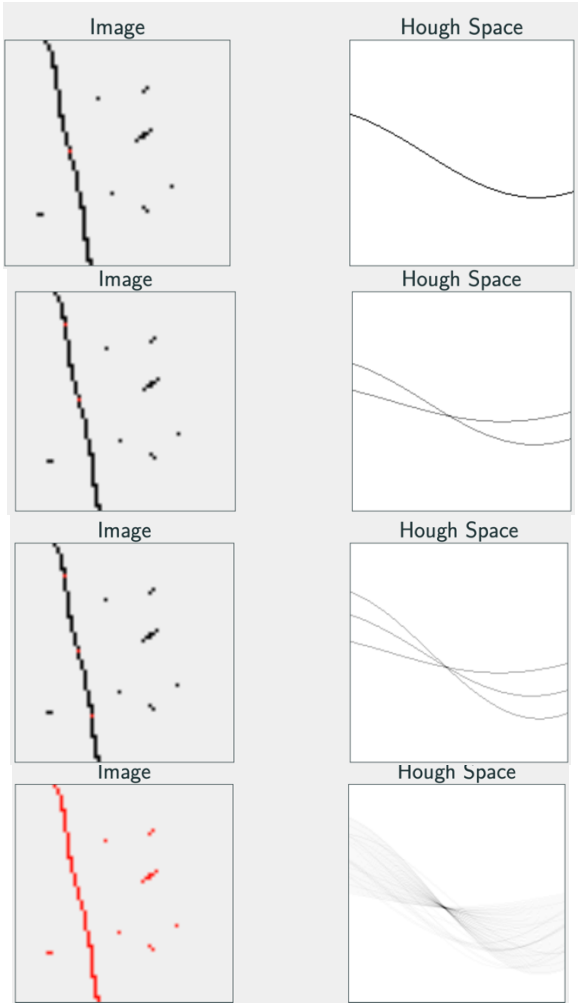
Then, we can discretize all possible line within an image on a 2D grid:

- θ is between 0 and 2π.
- r is between 0 and the length of the image diagonal.

Idea: Let points vote on which line is the best.

- Each point can be part of (infinitively) many lines.
- All potential lines going through this point are of interest
- Each point votes on all lines that go through it
  - This corresponds to a line in Hough space.
- Repeat for all points

Example: Hough Transform
Each point represents a line in the Hough Space



- A peak in Hough space corresponds to a line in the image.
- The place where they intersect is the line in the image where goes though all points.
- Can be found using non-maximum suppression.

Generalized Hough Transform

- We can generalize the Hough transform for more complex models.
  - E.g. for circles, the Hough space is now in 3D and each point becomes a conic.
- Hough space has the same number of dimensions as the model we fit has degrees of freedom.
- Impractical for more than three degrees of freedom.

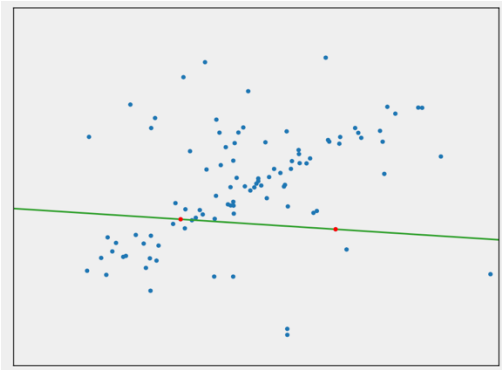
RANSAC

(Random sample consensus= RANSAC)
Idea: Instead of computing the Hough space, what if we could sample points directly in Hough space?

- What if we could sample with the value in Hough space being proportional to the probability of sampling the point?

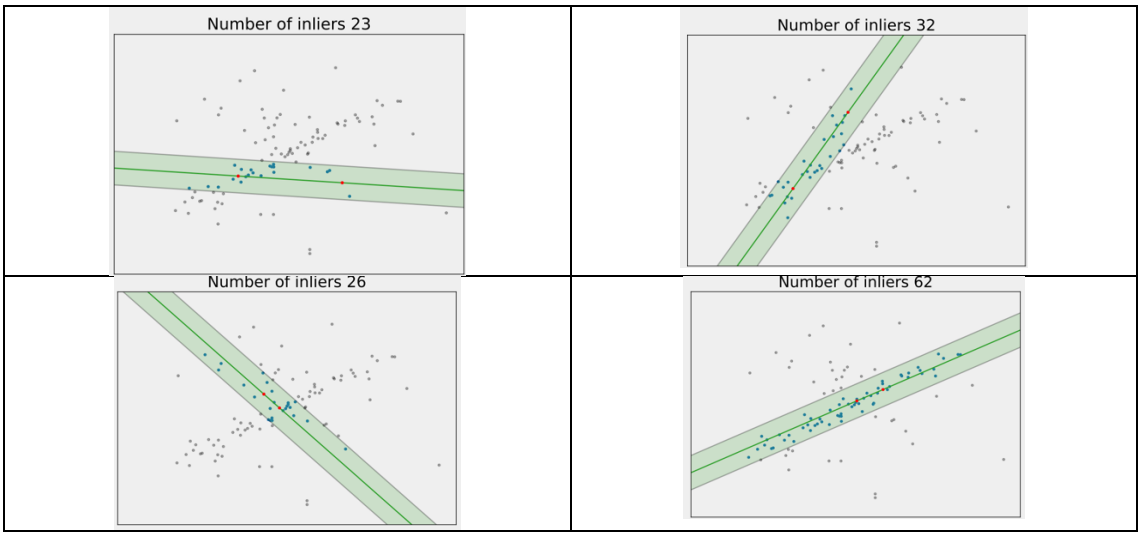
- Randomly sample the minimum number of points we need to fit our model.
- Fit the model to these samples.

Does this line fit the data well?
We need two points to fit a line through them.



Measure inliers:

Points closer than a certain threshold to the line are inliers!
Number of inliers is indication of how well the line fits.



We need to keep track which the line that gives the maximum number of inliers. If in the next iteration, it is not better, we don't care about it.

The RANSAC algorithm:
Keep track of which line that the most inliers so far, and update this if a line with more inliers are found.

- Sample minimum number of points required to fit model. → Fit model to these.
- Data points with an error less than τ are inliers with respect to the fitted model.
  - If number of inliers is higher than the highest number of inliers seen so far, update best model.
- Repeat for N iterations.
- Final step: Re-fit model to all inliers of the best model.

Implementation details:

- Represent lines with homogeneous coordinates
  - Makes it easy to compute distance to line
  - The first two coordinates should have norm 1.
- Recall that distance to line is given by: $[l \cdot \Pi^{-1}(p)\,]$

RANSAC:

- Sample in Hough space without computing it.
- Useful for fitting models when outliers are present.
- We must select the threshold for inliers and the number of iterations carefully.
- Being able to fit a model to the least amount of data points is interest.

How many iterations?

- We can compute with some idea of how many iterations we need.
- Assume fractions of outlies is ε, for example $\varepsilon = 0.1$. (fraction of outliers)
- We need n data points to fit a single model

$$\text{P (one sample has only inliers)} = (1-\varepsilon)^n$$
$$\text{P (one sample has outliers)} = 1-(1-\varepsilon)^n$$
$$\text{P (each of N samples has outliers)} = (1-(1-\varepsilon)^n)^N$$
$$\text{P(at least one of N samples has only inliers)} = 1-(1-(1-\varepsilon)^n)^N = p$$
$$p \Leftrightarrow N = \frac{log\,(1-p)}{log\,((1-(1-\varepsilon)^n))}$$

- Set p to a high value such as $p = 0.99$. Useful if we know ε. (usually it is not the case).

→ Then, we can determine the number of iterations adaptatively:

- We can estimate an upper bound of ε while running RANSAC.
- Let be s the largest number of inliers seen after m iterations.
- $\widehat{\epsilon} = 1 - \frac{s}{m}$
- We can now estimate an upper bound on the number of iterations required.
- $\widehat{N} = \frac{log\,(1-p)}{log\,((1-(1-\varepsilon)^n))}$
- Terminate once $m > \widehat{N}$

Week 8: Transform invariant features

Similarity:
Pixel-wise comparison: shift of single pixel vs. two views

Basic idea:

- locally appearance between views is the same
- Variation can be handled via invariances

SIFT – key elements:

- Features localized at interest points.
- Adapted to scale and invariant to appearance changes.

SIFT – scale invariant feature transform

Scale-space blob detection – difference of Gaussians

Harris corners and BLOBs

Harris corners are features that have a large change of intensity in two orthogonal directions. They are:

- Local
- Can be found at different scales by changing gaussian filters.
- Work in rotated frames.

Harris corners are found by first order derivatives whereas blobs are response to second order image derivatives.

BLOBs: binary large objects
Correspond to:

- Dark area surrounded by brighter intensities.
- Bright area surrounded by darker intensities.

Hessian:
The Hessian matrix contains the second order derivatives

$$H(x,y) = \begin{bmatrix} I_{xx}(x,y) & I_{xy}(x,y) \\ I_{xy}(x,y) & I_{yy}(x,y) \end{bmatrix}$$

Where

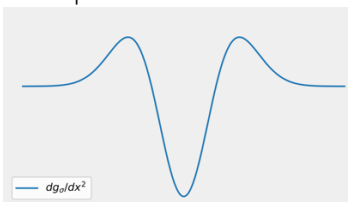
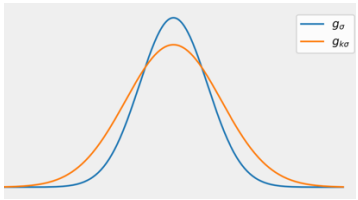
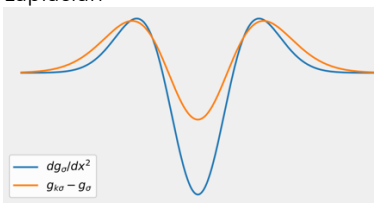
$$I_{xx}(x,y) = \frac{\delta^2 I(x,y)}{\delta x^2},\ I_{yy}(x,y) = \frac{\delta^2 I(x,y)}{\delta y^2}\ and\ I_{xy}(x,y) = \frac{\delta^2 I(x,y)}{\delta x \delta y}$$

Curvature:
Second order derivatives measure curvature.
Eigenvalues (λ_1, λ_2) measure the principal curvature, i.e., degree of change in intensity.
Eigenvectors measure direction of change where the eigenvector corresponding to the largest eigenvalue (λ_1) is the direction of most change and the second is orthogonal to that.

BLOB detection with Hessian:
Similar to Harris corner detector, we can use either of the measures

$$det(H) = \lambda_1\lambda_2,$$
$$trace(H) = \lambda_1+\lambda_2$$

where λ_i are the eigenvalues of the Hessian.
$det(H)$ is the Gaussian curvature.
$trace(H) = \nabla^2 I$ is the Laplacian, which we use for blob detection.

The Laplacian	Two Gaussians with different variance	Different of gaussians vs. Laplacian
		

BLOB detection DoG:
An approximation of $trace(H)$ is the Difference-of-Gaussian (DoG).
Consider the following convolution with two different Gaussian kernels:

$$\nabla^2 I \approx D_\sigma = (G_{k\sigma} - G_\sigma) * I = L_{k\sigma} L_\sigma$$

where G_σ is a Gaussian convolution with a width of σ and $k > 1$ is a scale factor.

SIFT – Scale invariant

Using difference of Gaussians for blob detection

$$D(x, y, \sigma) = ((G_{k\sigma} - G_\sigma) * I)(x, y) = L(x, y, k\sigma) - L(x, y, \sigma)$$

We need to ensure that DoG is scale normalized (the DoG kernel).

$$\sigma^2 \nabla^2 G$$

If we take an offset in the heat equation, we have

$$\sigma \nabla^2 G = \frac{\partial G}{\partial \sigma} \approx \frac{G_{k\sigma} - G_\sigma}{k\sigma - \sigma}$$

Multiplying both sides with σ we get:

$$\sigma^2 \nabla^2 G \approx \frac{\sigma}{k\sigma - \sigma}(G_{k\sigma} - G_\sigma) = \frac{1}{k-1}(G_{k\sigma} - G_\sigma)$$

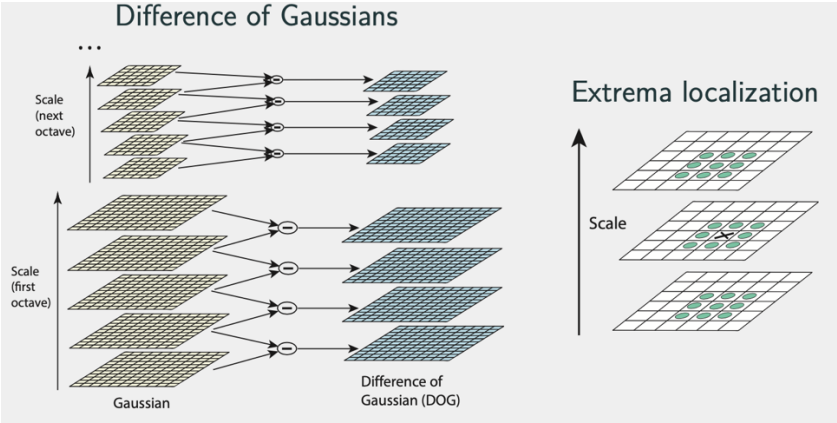
From this we get: $(k-1)\,\sigma^2 \nabla^2 G \approx G_{k\sigma} - G_\sigma$

Since k is constant over scales it does not affect the relative response of the DoG kernel.

Gaussian scale space – Efficient:

- Convolution of two Gaussians yield a new Gaussian
- Generate scale space by iteratively blurring already blurred images more (It requires smaller Gaussian kernels)
- $k = 2^{\frac{1}{3}}$
- σ doubles after three images, the image is down sampled. This is an octave.

SIFT – Estimation of DoG:



SIFT: Interest point localization

- Taylor approximation to second degree of 2D surface

$$D(x) = D + \frac{\partial D^T}{\partial x}x + \frac{1}{2}x^T\frac{\partial^2 D}{\partial x^2}x$$

- Setting the derivative of $D(x)$ to zero

$$\hat{x} = -\frac{\partial^2 D^{-1}}{\partial x^2}\frac{\partial D}{\partial x}$$

We get:

$$D(\widehat{x}) = D + \frac{1}{2}\frac{\partial D^T}{\partial x}\hat{x}$$

$|D(\widehat{x})| > 0.03$ otherwise, the point is discarded.

Interest point along edges discarded:

- The eigenvalues of the Hessian are proportional to the principal curvatures

$$H = \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix} = \begin{bmatrix} \frac{\partial^2 I}{\partial x^2} & \frac{\partial^2 I}{\partial x \partial y} \\ \frac{\partial^2 I}{\partial x \partial y} & \frac{\partial^2 I}{\partial y^2} \end{bmatrix}$$

- $\lambda_1 \; and \; \lambda_2$ are the eigenvalues of the Hessian

$$trace(H) = I_{xx} + I_{yy} = \alpha + \beta$$
$$det(H) = I_{xx}I_{yy} - I_{xy}^2 \; \alpha\beta$$

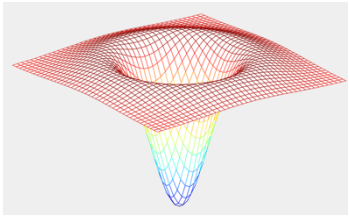
points are kept if

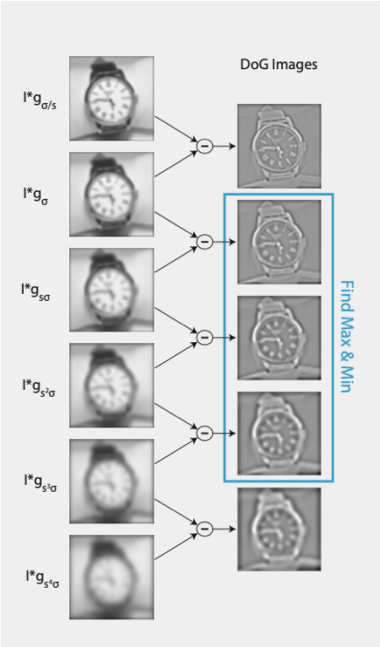
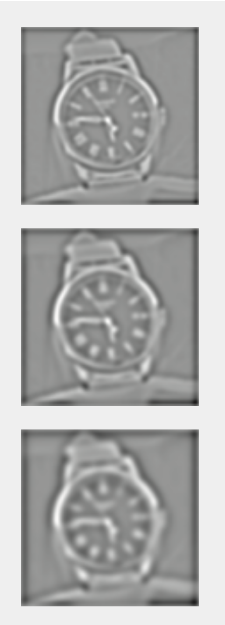
$$\frac{trace(H)^2}{det(H)} < \frac{(r+1)^2}{r}$$

Where r=10 (found to be a good heuristic)

DoG measure:

Features have |DoG(I)|>τ , where τ is a threshold.



DoG in scale pyramids:	Scale space blobs and DoG:
Scale pyramids are smaller scaled versions of the same image. Scaled DoG is subtraction between all layers.	DoG at different scales makes for a scale invariant feature detector. Small and large details are recoverable in different DoGs.
	

SIFT: Orientation assignment

Compute the orientation of gradients in a small region around the BLOB.

$$m(x,y) = \sqrt{L_x^2 + L_y^2}$$
$$\theta(x,y) = arctan2(L_y, L_x)$$

Where

$$L_y = L(x+1,y) - L(x-1,y)$$
$$L_x = L(x,y+1) - L(x,y-1)$$

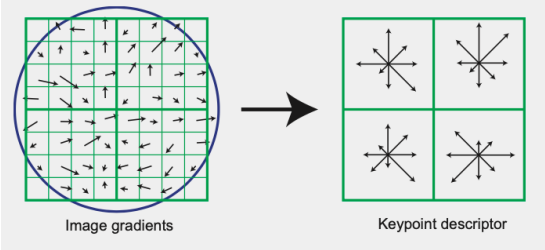
- Compute circular histogram of gradient orientations. Weighted by magnitude, smoothed, and has 36 bins.
- Use peak in histograms to assign orientation of point
- This introduces rotation invariance
- Can we multiply peaks in histogram?
  - Yes, this can happen at e.g., corners
  - Create a new point at the same location if peak is over 80% of max.

SIFT: Invariances

- Position
- Scale
- Rotation
- Linear intensity change
- Perspective change?

SIFT: Interest point descriptor

- Create local patch at scale and orientation of point
- Build a histogram of local gradient orientations



- Normalized using L_2 norm: $d_n = \frac{1}{\sqrt{\sum_{i=1}^{128} d(i)^2}} d$

Matching of descriptors:

Taylor SIFT

- Use Euclidean distance between normalized vectors:

$$\delta(d_i, d_j) = \sqrt{\sum_{n=1}^{128} (d_{i,n} - d_{j,n})^2}$$

- Note – for comparison the square root is not needed

- RootSIFT: Simple trick to improve SIFT matching
  - SIFT is a histogram
  - Euclidean distance is dominated by large values
  - RootSIFT is a transformation that measures distance using the Hellinger kernel.
    - L1 normalize.
    - Take the square root of each element.
    - L2 normalize the resulting vector.
  - Compare using Euclidean distance.

- For each feature in image 1($d_{1,i}$) find the closest feature in image 2 ($d_{2,j}$). This will give a lot of incorrect matches.

- Cross checking
  - Only keep match where $(d_{2,j})$ is also the closest to $(d_{1,i})$ of all features in image 1.

- Ratio test
  - Compute the ration between the closest and second closest match, and keep where this is below a threshold, e.g., 0.7.

SIFT: SUMMARY

- SIFT is both a feature detector and descriptor
- Find local extrema of DoGs in scale space
- Place patch oriented od DoGs in scale space
- Compute histograms of gradients.
- Allows matching of images invariant to scale, rotation, illumination, and viewpoint.
- Partly visible objects can be matched.

Other descriptors:

SIFT is widely used.

- ORB
- SURF
- BRISK
- FAST
-

Learned descriptors:

- Deep Learning has created improved feature descriptors.
- Mostly in improvement in invariance to changing lighting.

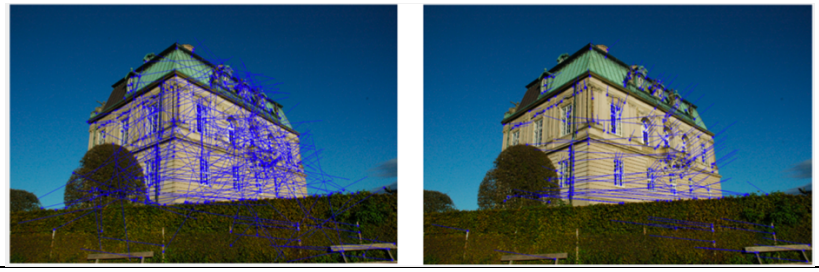
Week 9: Geometry Constrained Feature Matching

Setting the scene:

- Stereo geometry
- SIFT features
- RANSAC

Geometry constrained feature matching:

- Use multi view geometry to filter matches



Recap fundamental matrix:

R and t describe the relative pose between the cameras.

$$E = [t]_x R$$
$$F = K_2^{-T} E K_1^{-1}$$
$$0 = q_2^T F q_1$$

The fundamental matrix expresses those corresponding points lie on their epipolar lines.

Estimating the fundamental matrix

Fundamental matrix problem:

The fundamental matrix is defined with the relation $q_2^T F q_1 = 0$.

Consider the points q_{1i} and q_{2i} projected into cameras one and two, respectively. The relation is then:

$$0 = q_{2i}^T F q_{1i}$$
$$= \begin{bmatrix} x_{2i} \\ y_{2i} \\ 1 \end{bmatrix}^T \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{bmatrix} \begin{bmatrix} x_{1i} \\ y_{1i} \\ 1 \end{bmatrix}$$

Rearrange the terms:

$$0 = q_{2i}^T F q_{1i}$$
$$0 = B^{(i)}\, flatten(F^T)$$

Where

$$B^{(i)} = [x_{1i}x_{2i} \quad x_{1i}y_{2i} \quad x_{1i} \quad y_{1i}x_{2i} \quad y_{1i}y_{2i} \quad y_{1i} \quad x_{2i} \quad y_{2i} \quad 1]$$
$$vec(F) = [F_{11} \quad F_{21} \quad F_{31} \quad F_{12} \quad F_{22} \quad F_{32} \quad F_{13} \quad F_{23} \quad F_{33}]^T$$

Fundamental matrix solution:

Define B

$$B = \begin{bmatrix} B^{(1)} \\ B^{(2)} \\ \dots \\ B^{(n)} \end{bmatrix}$$

Subject to $\|flatten(F^T)\|_2 = 1$ the solution is the singular vector with the smallest singular value.

Linear algorithm

Degrees of freedom: **Eight-point algorithm:**
F has 9 numbers and is scale invariant.
Each pair of corresponding point fixes a degree of freedom. Eight points is enough to estimate the fundamental matrix.
This is the eight-point algorithm.

Degrees of freedom: **Seven-point algorithm:**
$[t]_x$ has rank 2 and thus, F is also rank deficient, i.e., $det(F) = 0$
Thus, F has seven degrees of freedom, and can be found from 7 matches.
B will have two singular vectors with singular value 0.
Denote these F' and F^+.
$F = F' + (1 - \alpha)F^+$, where α is chosen such that $det(F) = 0$.
This is the seven-point algorithm.

→ F can be estimated from eight-point correspondences easily.
→ Possible to estimate from just seven. Can be followed by non-linear optimization.

Incorporating RANSAC

To use RANSAC, we need a way to measure distance from our model.
What does $q_{2i}^T F q_{1i}$ equals if points are not corresponding?
$q_{2i}^T F$ and $F q_{1i}$ are epipolar lines.

Distance from point to line:

$$d = [a \quad b \quad c] \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}^T$$

Should have $a^2 + b^2 = 1$, but is not fulfilled for the epipolar lines.

Geometric distance:	Sampson's distance:
We can normalize the distance to both epipolar lines, using their two coordinates. The squared geometric distance is then given by $= (q_{2i}^T F q_{1i})^2 \frac{1}{(q_{2i}^T F)_1^2 + (q_{2i}^T F)_2^2} \frac{1}{(F q_{1i})_1^2 + (F q_{1i})_2^2}$ Where x_i^2 refers to the square of the i[th] element of x.	A similar distance is Sampson's distance. $d_{samp}(F, q_{1i}, q_{2i}) = \frac{(q_{2i}^T F q_{1i})^2}{(q_{2i}^T F)_1^2 + (q_{2i}^T F)_2^2 + (F q_{1i})_1^2 + (F q_{1i})_2^2}$ Performs slightly better than the geometric distance in practice. Is a squared distance.

Thresholding for RANSAC

How to choose the threshold for RANSAC?

Thresholding distances:
- How to choose the threshold for RANSAC?
- Introduce assumptions.
- Assume that the errors of inliers follow a normal distribution.

Dimensionality of the error:
- Fundamental/essential matrix:
  - The error is the distance to the epipolar line.
  - This is an error in one dimension.
- Homography
  - The error is the distance from mapped point to true point.

    - This is an error in two dimensions.
- Pose estimation/camera calibration
    - Again, it is a distance between points.
    - This is an error in two dimensions.

This is called the *codimension* of the problem. We denote this m.

Choosing the threshold:

- Assume that the error of each sample follows an m-dimensional normal distribution with *standard deviation* σ.
- The squared error is $X_m^2 \; distributed \; (by \; definition)$.
- Always work with squared distances.
    - Not necessary to take square root when comparing distances.
- Choosing a confidence interval e.g., 95%.
    - i.e., we want our threshold to correctly identify 95% of all inliers.
- Look up the CDF for our X_m^2 distribution.
- E.g., for a fundamental matrix and 95%, $\tau^2 = 3.84 \cdot \sigma^2$

Some values from the CDF of X_m^2:

$m \backslash 1-p$	90%	95%	99%
1	2.71	3.84	6.63
2	4.61	5.99	9.21

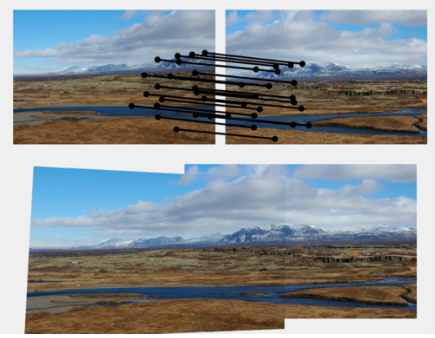
Week 10: Image Stitching

Using:

- Homographies (hest)
- (SIFT) features
- RANSAC

Image stitching

Panorama:



About homographies:

- What does a homography describe?
  The mapping of points between to images viewing the same plane
- BUT! The mountains above are not a plane. Why can we still use homography?

Homographies for panoramas:

- When the camera does not move but only rotates there are no perspective deformations.
- It is equivalent to looking at a painting of the world.
- Therefore, we can assume the world is flat and use a homography.
- But what happens if the camera moves?

Finding inliers

Measuring the error of a match:

Let q_1 and q_2 be matching points without errors, such that:

$$p_1 = Hp_2$$
$$p_1 = \Pi^{-1}(q_1)\,, \quad p_2 = \Pi^{-1}(q_2)$$
$$q_1 = \Pi(\mathrm{H}\Pi^{-1}(q_2))$$

We have some 2D error in our detection of the points.

$$\tilde{q}_1 = q_1 + \epsilon_1\,, \quad \tilde{q}_2 = q_2 + \epsilon_2$$
$$\tilde{p}_1 = \Pi^{-1}(\tilde{q}_1)\,, \quad \tilde{p}_2 = \Pi^{-1}(\tilde{q}_2)$$

The error is the distance to the observed point in both images.

$$\|\tilde{q}_1 - \Pi(\mathrm{H}\Pi^{-1}(q_2))\|_2^2 + \|\tilde{q}_2 - q_2\|_2^2$$

Do we know $\boldsymbol{q_2}$? Not!

We can use optimization to estimate the error free point.

$$dist_{true}^2 = \min_{q_2} \|\tilde{q}_1 - \Pi(\mathrm{H}\Pi^{-1}(q_2))\|_2^2 + \|\tilde{q}_2 - q_2\|_2^2$$

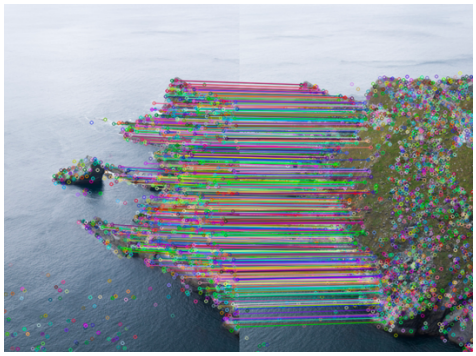
Requires soling a least squares problem for each distance computation. Impractical!

Practical approximation:

Map the observed points back and forth and compute the distance.
Use the distance to measure if a pair of points are inliers with respect to a homography.

Transforming images

Example: Images and matches



How to use the homography?

- We have estimated the homography.
- How do we use it?
- Similarity to image undistortion.

- We can use the homogrpahy to warp an image to our current field of view.
- Generate all x, y coordinates for all pixels in the reference image.
- Map these to the other image using the homogrpahy.
- Use bilinear interpolation to compute the value at the transformed pixel locations.

Warping:

- You have to decide in which are to evaluate the homography.
- Simple, evaluate all homographies in the same area
  - Wasteful computation
  - Simple to implement
- Warp only the valid part of each image and move the images around.
  - Necessary for larger panoramas.

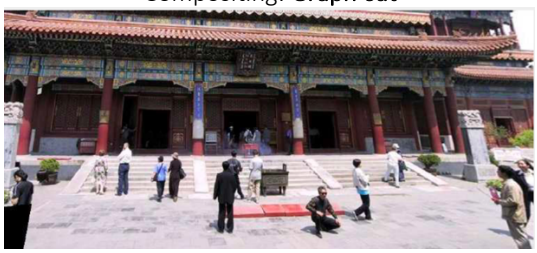
Example: Warped to same coordinate system

Note that x is negative.

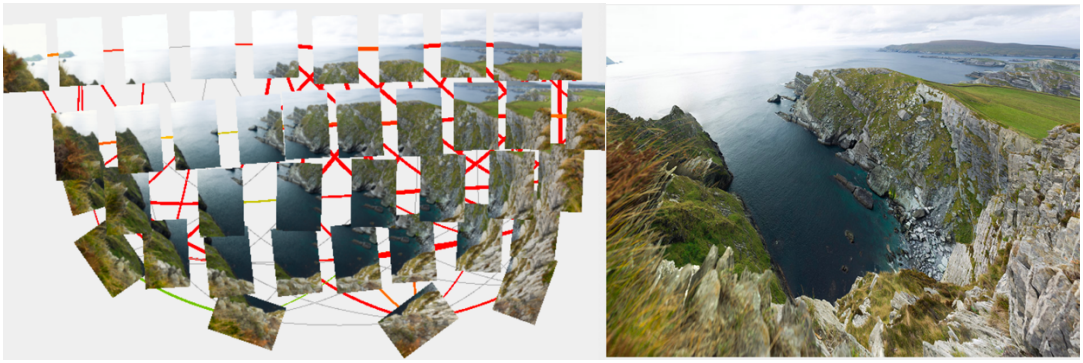
Compositing images:

- **Overlap**
- **Median**
- **Average**
- **Graph cut**

Example: Composited

Compositing: **Average**	Compositing: **Graph cut**
	

Multiple images



How do we handle more than two images?

Example with three images:

Find the homographies between each pair $H_{1\to2}$ and $H_{2\to3}$.

Product of homographies are new homographies.

$$H_{1\to3} = H_{1\to2}H_{2\to3}$$

This principle extends to as many images as desired.

Multiple images:

- Choose compositing surface
  - I.e., which image is our baseline.
  - Easy option is to choose one image to be the center.
- Joint optimization

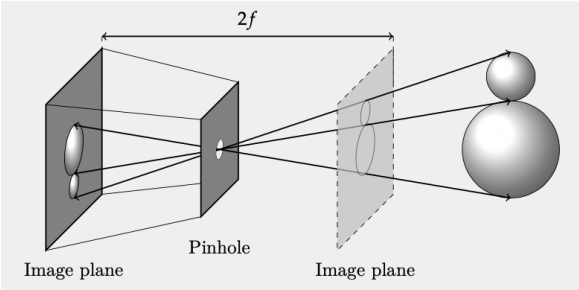
Nodal point

Capturing your own images:

Which point should you actually rotate around?

- Image sensor?
- Pinhole?

We should rotate the pinhole since is where all points intersect.



But in the real-world camera scenario, which point should you actually rotate around?

→ It depends on the camera, usually in the middle of the lens.

What happens if we rotate around the wrong point?

→ We get artifacts.

And more!

- Known focal length reduces degree of freedom
- Other projection models
- Exposure/color correction

Week 11: Visual Odometry

Motivation:
Figure out how you are moving though the world by using a camera has many applications:

- Drones
- Robotic vacuum cleaners
- Virtual reality headsets
- Augmented reality apps
- Autonomous cars

Many similar and related concepts:

- Visual Odometry
- SLAM (Simultaneous Localization and Mapping)
- SfM (Strucutre from motion)

They all deal with some form of estimating a 3D map of the world and camera poses, but have emphasis on different parts.

Multiple "cameras":

The unknown scale of t – Mathematical argument:

$$\mathrm{E} = \mathrm{R}[\mathrm{t}]_{\mathrm{x}}$$
$$0 = \mathrm{E}\,(\mathrm{st}) = \mathrm{R}[\mathrm{t}]_{\mathrm{x}}(\mathrm{st})$$

We can see that $\mathbf{t}$ lies in the null space of $\mathbf{E}$ but also that it can be arbitrarily scaled.

Decomposing the Essential Matrix

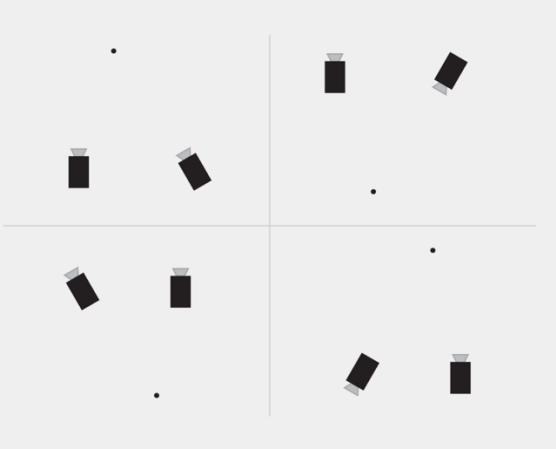
Essential matrix:
You have matched features between two cameras and want to make it robust. Estimate $\mathbf{F}$ or $\mathbf{E}$ with RANSAC.

Estimating E:

- How many points are required?
- Ask yourself:
  - How many degrees of freedom does it have?
  - How many degrees of freedom does a single point fix?
- Five
- Not possible with linear algorithm from five points.
- Typically estimated using Nister's five-point algorithm
  - Involves solving tenth degree polynomial
  - Is implemented in OpenCV.

Decomposing the Essential Matrix:

- The essential matrix can be computed from $\mathbf{R}$ or $\mathbf{t}$.
  - Can we recover them from $\mathbf{E}$?
- Decomposing the essential matrix is ill posed.
  - Two possible rotations.
  - The sign of the translation is unknown.
- A total of four possible poses for the second camera.
  - Check all four combinations.
  - Choose the one with the most points in front of both cameras.



Back to visual odometry:

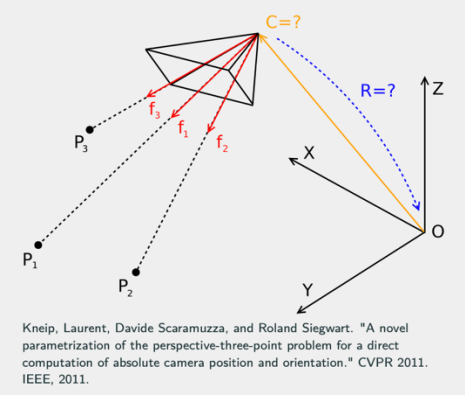
We can find the pose of two cameras relative to each other

- How can we estimate the pose of a third camera?
- Using the essential matrix again will give us a new arbitrarily scaled translation.
- **Idea:** Use the translation between the first two cameras to fix the scale.
- Triangulate points using the first two cameras
  - Use 3D points to find the pose of the third camera.

Perspective-n-Point

The Perspective-n-Point (PnP) problem.

→ Estimating the pose of a calibrated camera from no corresponding 3D-2D correspondences.



Kneip, Laurent, Davide Scaramuzza, and Roland Siegwart. "A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation." CVPR 2011. IEEE, 2011.

PnP vs camera resectioning:

In week 4 you did this for an uncalibrated camera (pest).

For an uncalibrated camera it is also called camera resectioning.

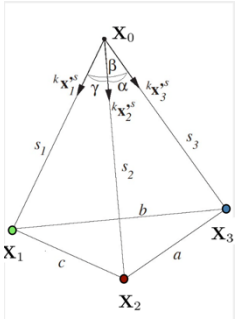
Naïve solution:

- Estimate the projection matrix ($\widetilde{P}$)
- Compute $K^{-1}\widetilde{P}$
- $K^{-1}\widetilde{P} \approx [R\ t]$
  - R is likely not a proper rotation matrix
  - Requires many points

→ How many points are required?

  - How many degrees of freedom does it have?
  - How many degrees of freedom does a single point fix?

- Three correspondences are required.
- This minimal case is therefore also known as P3P.



P3P - Geometry:

- The three 2D points give three pairwise angles.
- The distances three 3D points give three pairwise distances.

PnP rounding off:

- Three correspondences generate four possible solutions, and a fourth correspondence can be used to choose the correct one.
- Multiple algorithms exist and are implanted in OpenCV.
- Use RANSAC to make it robust.

Pose vs. position:

The pose of a camera is given by R and t.
This is not the orientation and position of the camera.

$$T_{world \to cam} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$$

$$T_{cam \to world} = T^{-1}_{cam \to world} = \begin{bmatrix} R^t & -R^t t \\ 0 & 1 \end{bmatrix}$$

- ➢ The orientation of a camera is given by R^t.
- ➢ The position of a camera is given by $-R^t\ t$.
- ➢ This is important in order to plot the camera.

Putting it all together

Outline:

1. Use the essential or fundamental matrix to estimate the pose of the second camera.
2. Triangulate points using the known camera poses.
3. Use PnP or camera resectioning to estimate the pose of the next camera.
4. Repeat steps 2 and 3.
5. Use (windowed) bundle adjustment.

Feature tracking:

- Some points can be tracked through many frames.
- Keep track of them to make your model drift less.