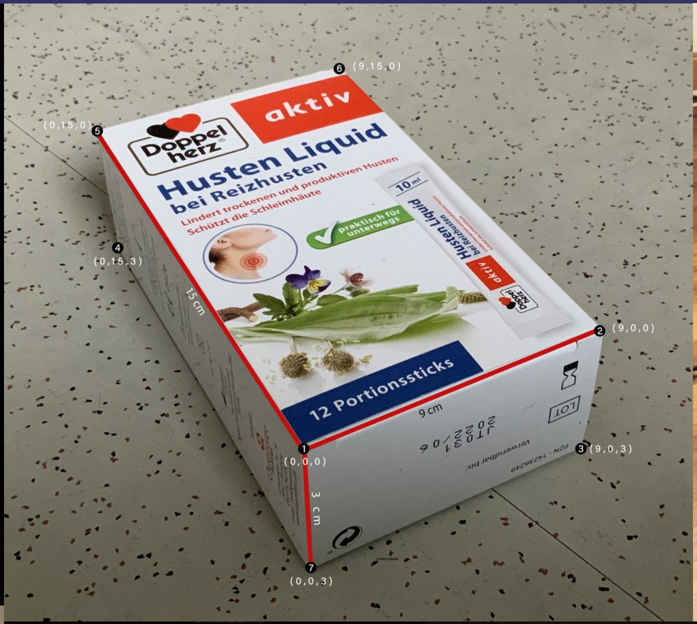
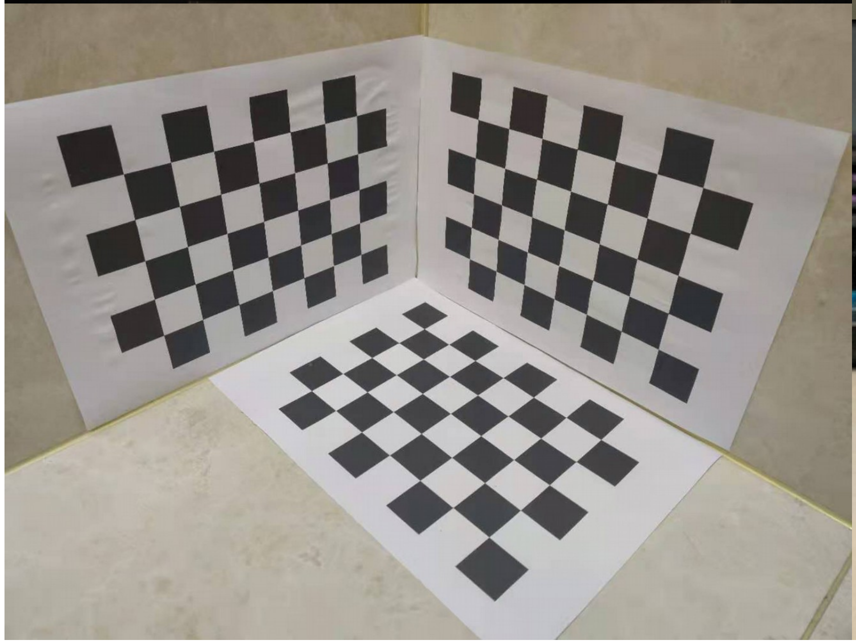
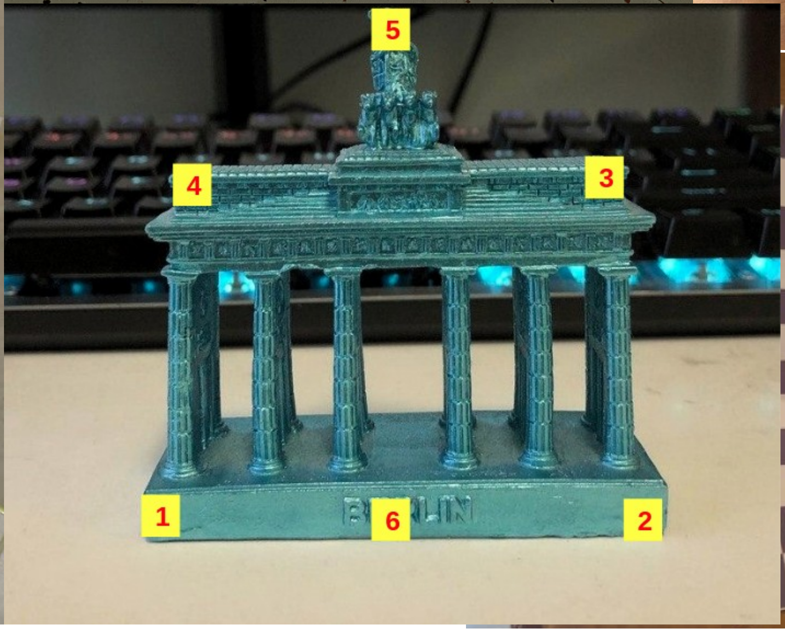
Photogrammetric Computer Vision

Berlin University of Technology (TUB),
Computer Vision and Remote Sensing Group
Berlin, Germany

Last Exercise





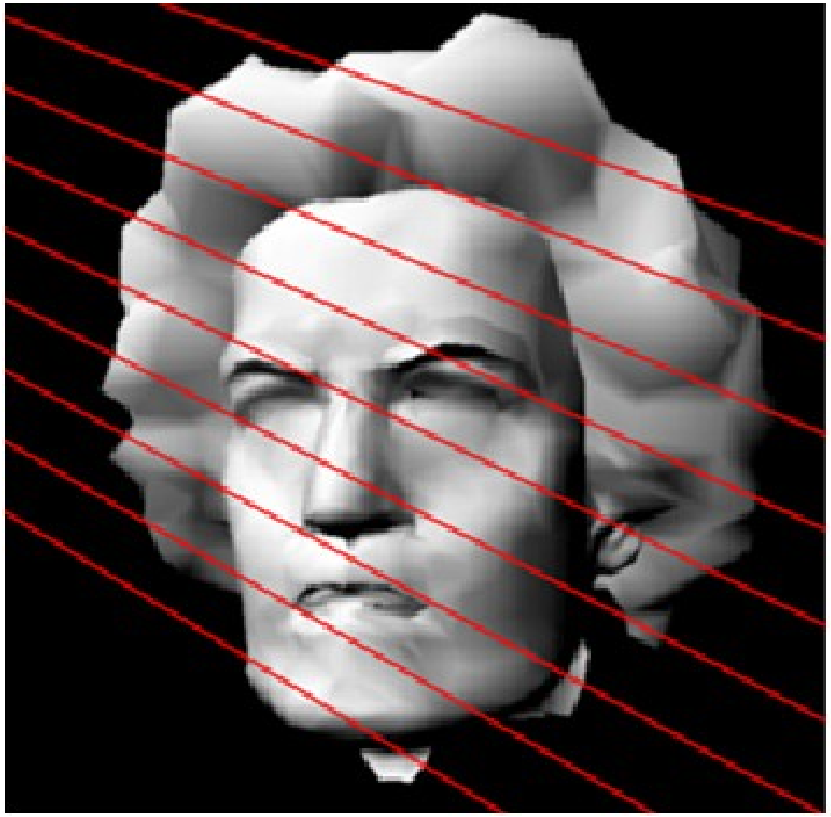


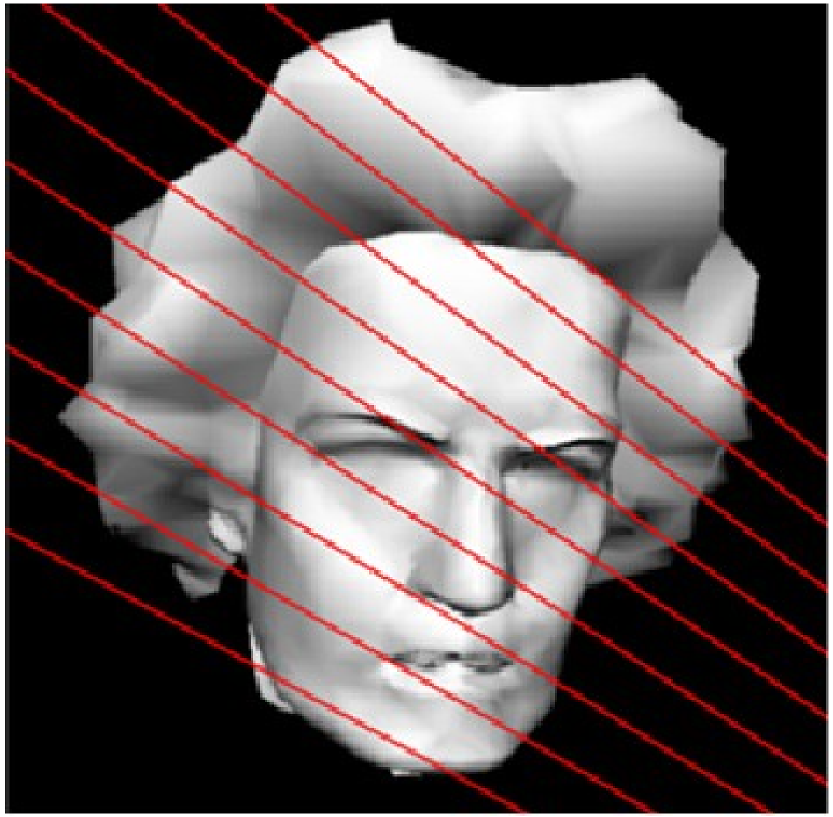
Relative Orientation

Orientation of an image pair

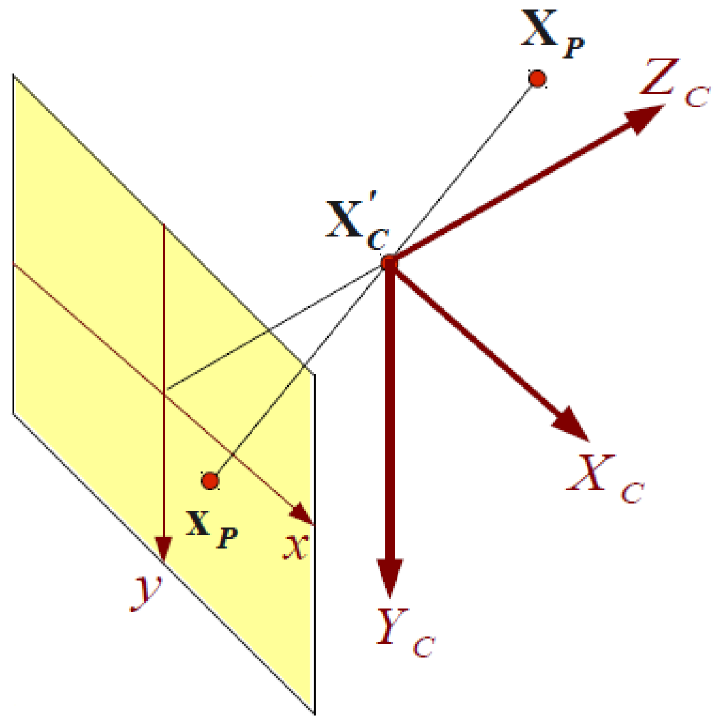
The relative orientation of an image pair is defined by the epipolar geometry.

Using algebraic projective geometry the epipolar geometry can be represented by the fundamental matrix.





What we have treated so far:



- Algebraic projective geometry
  - Homogeneous coordinates
- Object space coordinate system
- Camera coordinate system
- Image coordinate system
- Orientation of a single image
  - Projection matrix $\mathbf{P}$, calibration matrix **K**, projection center $\boldsymbol{X}_C$

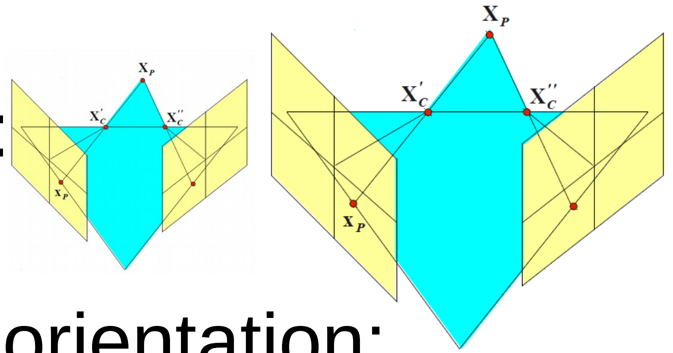
$$\mathbf{P} = \mathbf{K}\,\boldsymbol{R}\left(\boldsymbol{I} \middle| -\boldsymbol{X}_c\right)$$

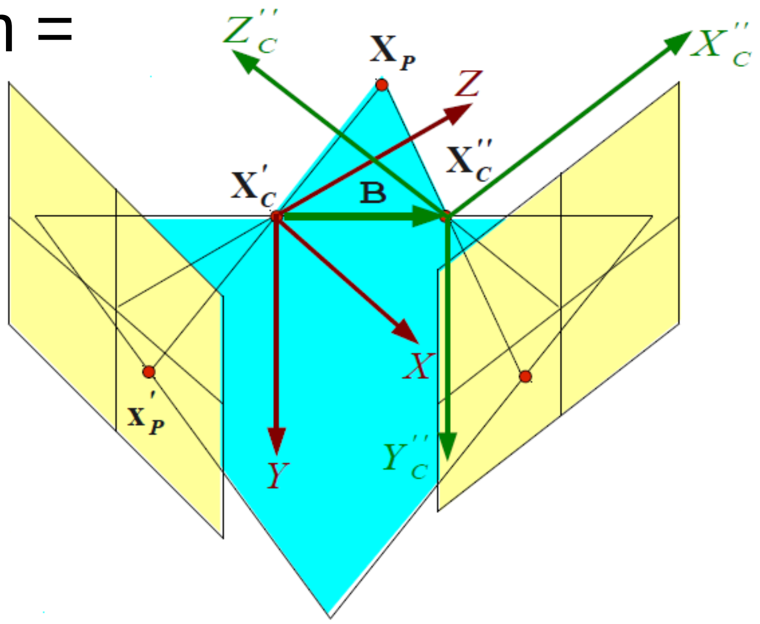
- Projection of object point $\mathbf{X}_P$ to image point $\mathbf{x}_P$:

$$\mathbf{x}_P = \mathbf{K}\,\boldsymbol{R}\left(\boldsymbol{I} \middle| -\boldsymbol{X}_c\right)\mathbf{X}_P = \mathbf{P}\,\mathbf{X}_P$$

What we want to achieve now:

- Derivation of relative orientation:
  - Where is the second camera w.r.t. the first one?
  - Into which direction does it look, relative to the first camera?
- Important: Invariance of the image pair:
  - Cameras do not determine scale!
- Selection of parametrization of relative orientation:
  - Setting: object space coordinate system = coordinate system of the first camera
- Wanted:
  - Baseline (direction): **B**
  - Rotation of the second camera $\boldsymbol{R}''$

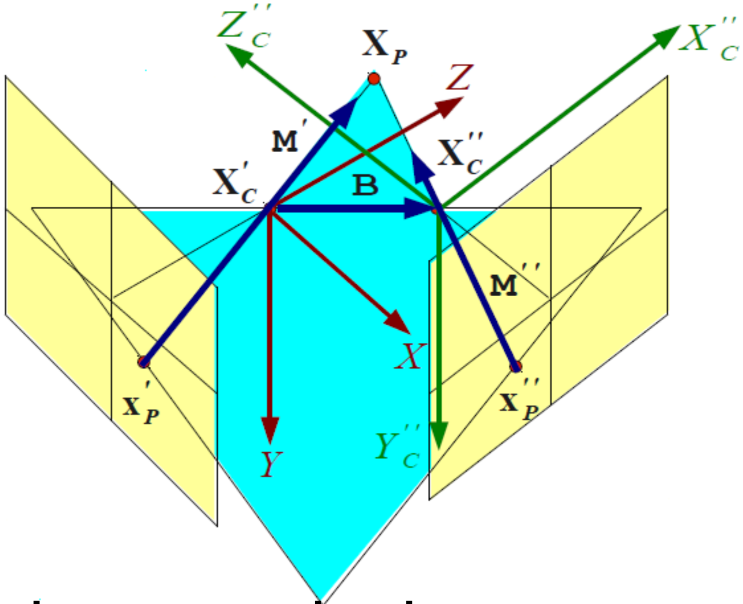




Coplanarity Condition

- If three 3D vectors are coplanar, the enclosed volume, i.e. the scalar triple product, is zero:

$$\mathtt{M}'' \cdot \left(\mathbf{B} \times \mathtt{M}'\right) = 0$$



- The three vectors must be given in the same coordinate system:
  - $\mathtt{M}'$ is direction from projection center to image point in **object space coordinate system**

$$\mathtt{M}' = (\boldsymbol{R}')^{-1} (\mathbf{K}')^{-1} \mathbf{x}'_P$$

  - As $\boldsymbol{R}'=\boldsymbol{I}$, for the first camera note e.g.:

$$\mathtt{M}' = \begin{bmatrix} 1/c & & \\ & 1/c & \\ & & 1 \end{bmatrix} \begin{bmatrix} x'_P \\ y'_P \\ 1 \end{bmatrix} = \begin{bmatrix} x'_P \\ y'_P \\ c \end{bmatrix}$$

  - analogously $\mathtt{M}'' = (\boldsymbol{R}'')^{-1} (\mathbf{K}'')^{-1} \mathbf{x}''_P$
  - Base line $\mathbf{B} = \boldsymbol{X}''_C - \boldsymbol{X}'_C$

Derivation of the General Formulation of the Coplanarity Condition

- We note $\mathbf{M}''\cdot(\mathbf{B}\times\mathbf{M}')=0=\mathbf{M}''^{\mathsf{T}}S_{\mathbf{B}}\mathbf{M}'$
  - with the skew symmetric matrix

$$S_{\mathbf{B}}=\begin{bmatrix}0 & -B_Z & B_Y\\ B_Z & 0 & -B_X\\ -B_Y & B_X & 0\end{bmatrix} \text{ where } \mathbf{B}=\begin{bmatrix}B_X\\ B_Y\\ B_Z\end{bmatrix}$$

- Insertion results in

$$\mathbf{x}_{P}''^{\mathrm{T}}(\mathbf{K}'')^{-\mathrm{T}}(\boldsymbol{R}'')^{-\mathrm{T}}\boldsymbol{S}_{\mathbf{B}}(\boldsymbol{R}')^{-1}(\mathbf{K}')^{-1}\mathbf{x}_{P}'=0$$

- With the **Fundamental Matrix**

$$\mathbf{F}=(\mathbf{K}'')^{-\mathrm{T}}\boldsymbol{R}''\boldsymbol{S}_{\mathbf{B}}\boldsymbol{R}'^{\mathrm{T}}(\mathbf{K}')^{-1}$$

we get

$$\mathbf{x}''^{\mathrm{T}}\mathbf{F}\mathbf{x}'=0$$

Coplanarity Condition of Calibrated Cameras

- If the calibration matrix in known, normalized image coordinates can be computed:

$$ {}^{n}\mathbf{x}' = (\mathbf{K}')^{-1}\mathbf{x}'_{P} $$

- With these coordinates as a specialized version of the fundamental matrix we get the Essential Matrix $\mathbf{E}$

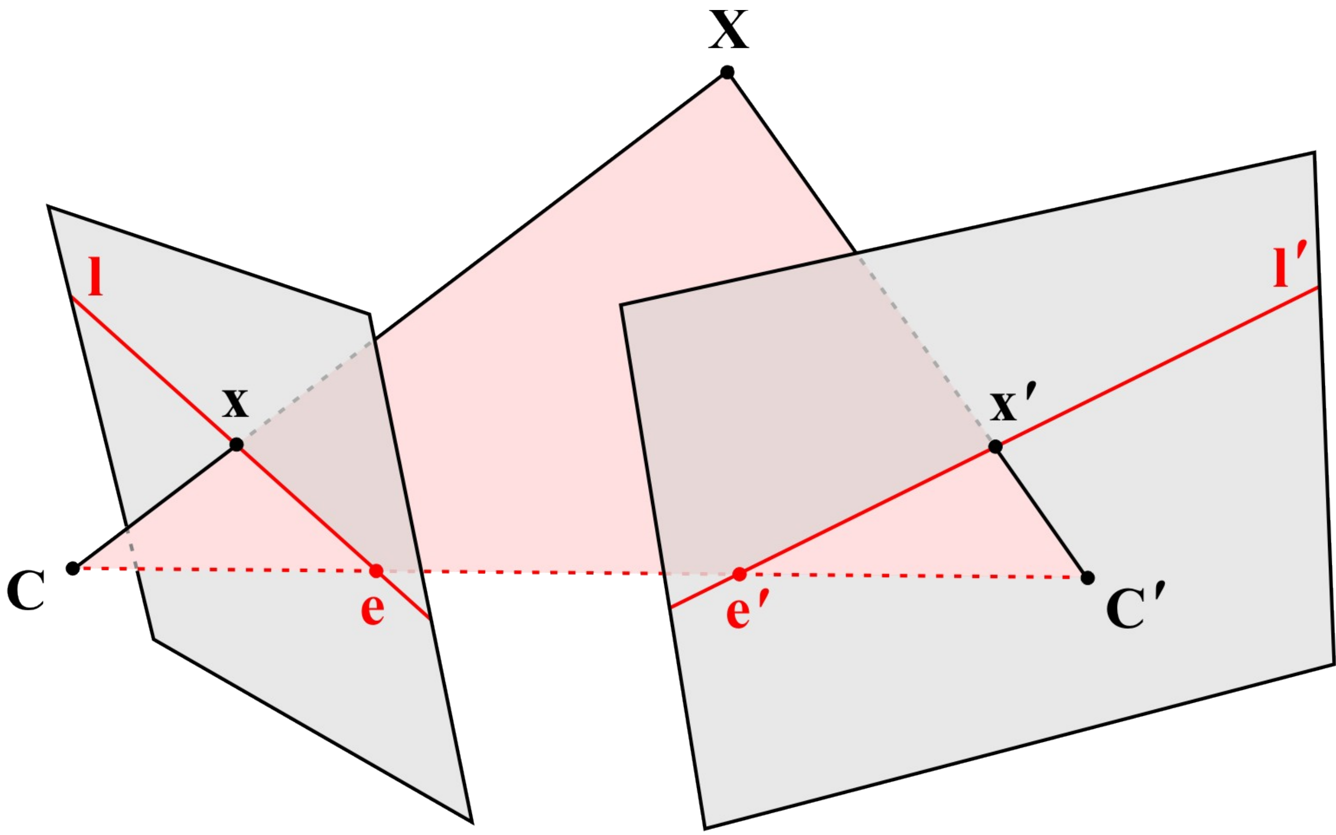
$$ \mathbf{E} = \boldsymbol{R}''\boldsymbol{S}_{\mathbf{B}}\boldsymbol{R}'^{\mathrm{T}} $$

- Then the coplanarity condition is noted as

$$ \mathbf{x}'^{\mathrm{T}}\mathbf{E}\,\mathbf{x}'' = 0 $$

Fundamental Matrix

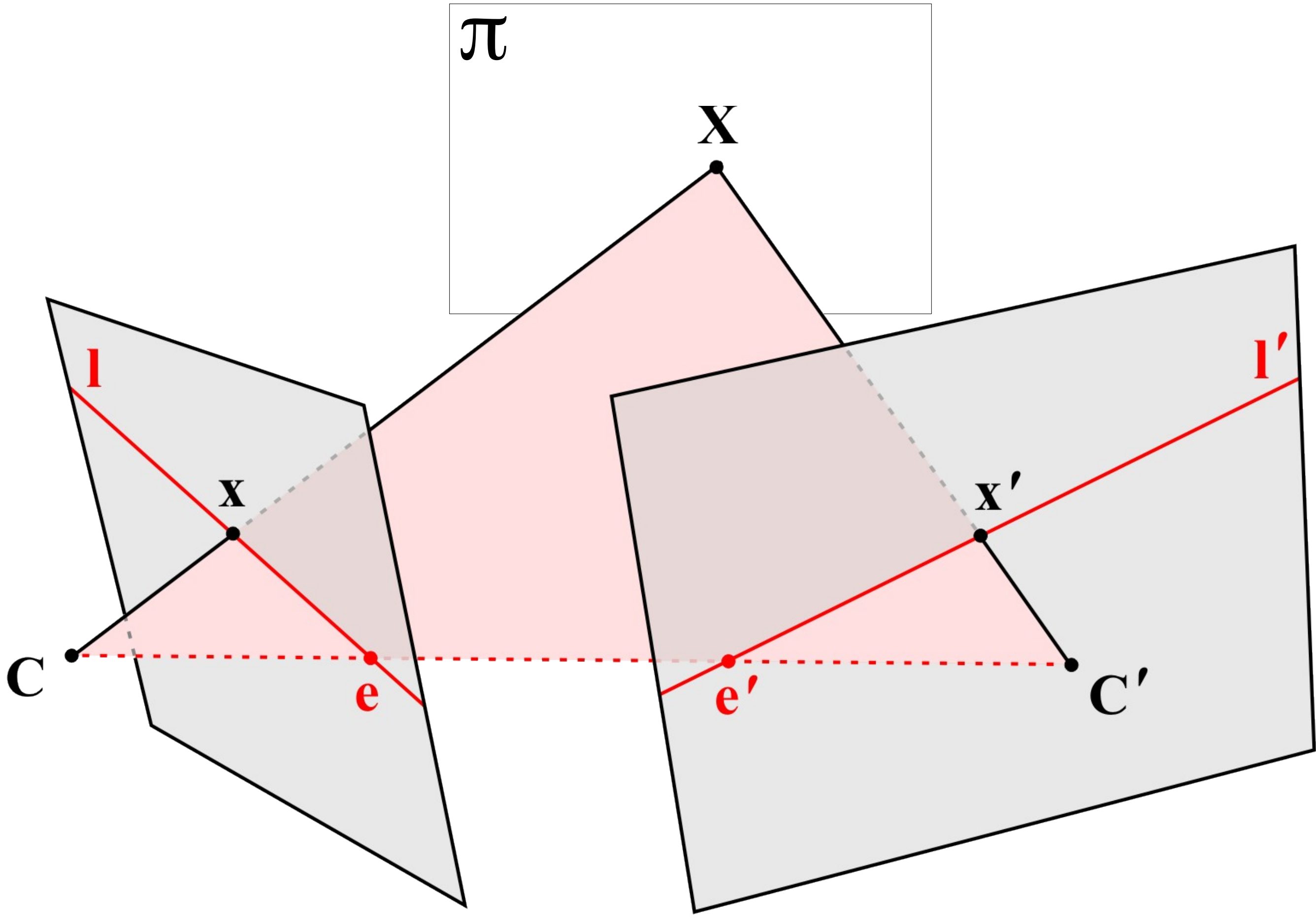
The fundamental matrix



Fundamental Matrix

The fundamental matrix

$$x' = H_\pi x$$

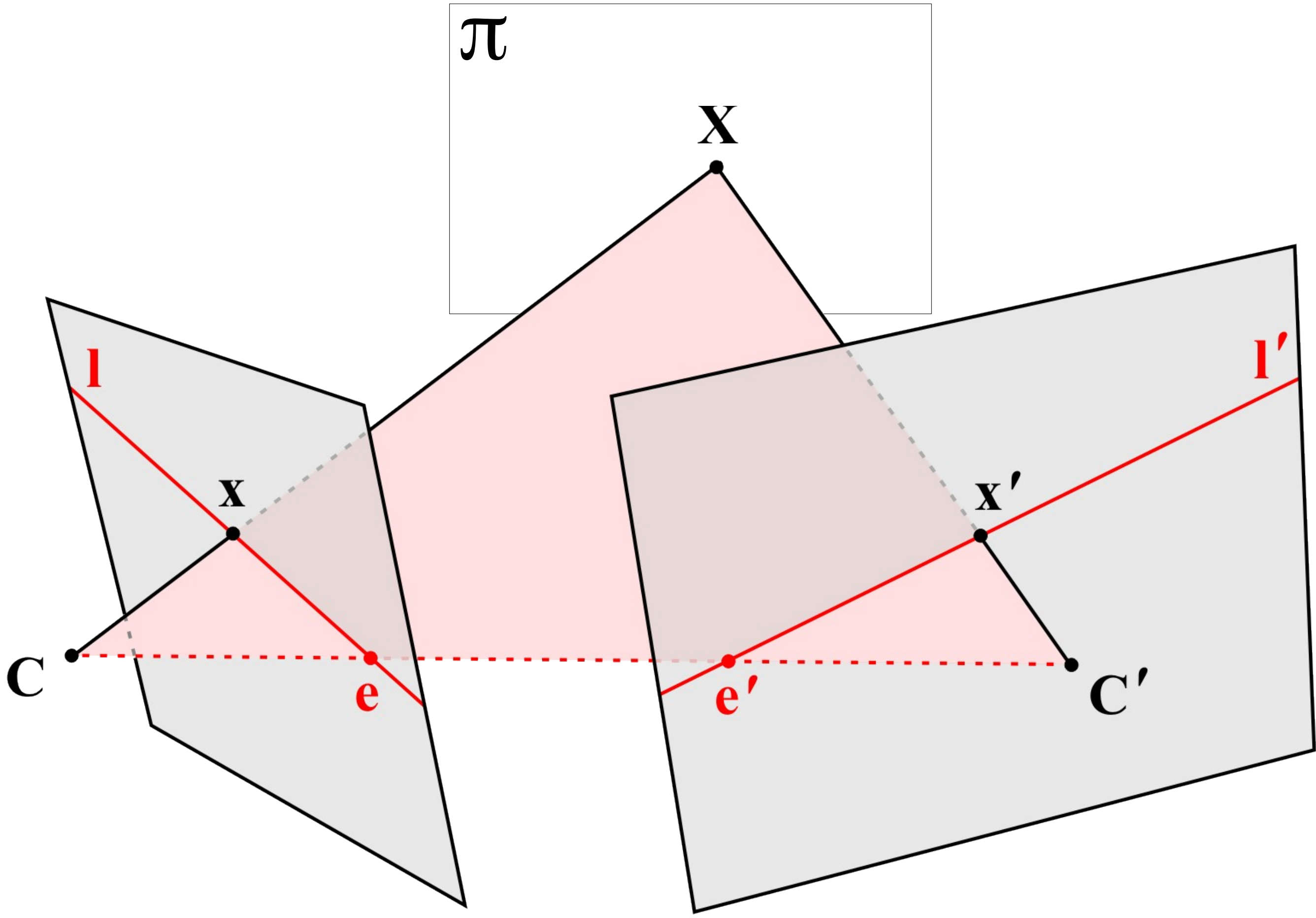


Fundamental Matrix

The fundamental matrix

$$x' = H_\pi x$$

$$\begin{aligned} l' &= e' \times x' \\ &= [e']_\times x' \end{aligned}$$

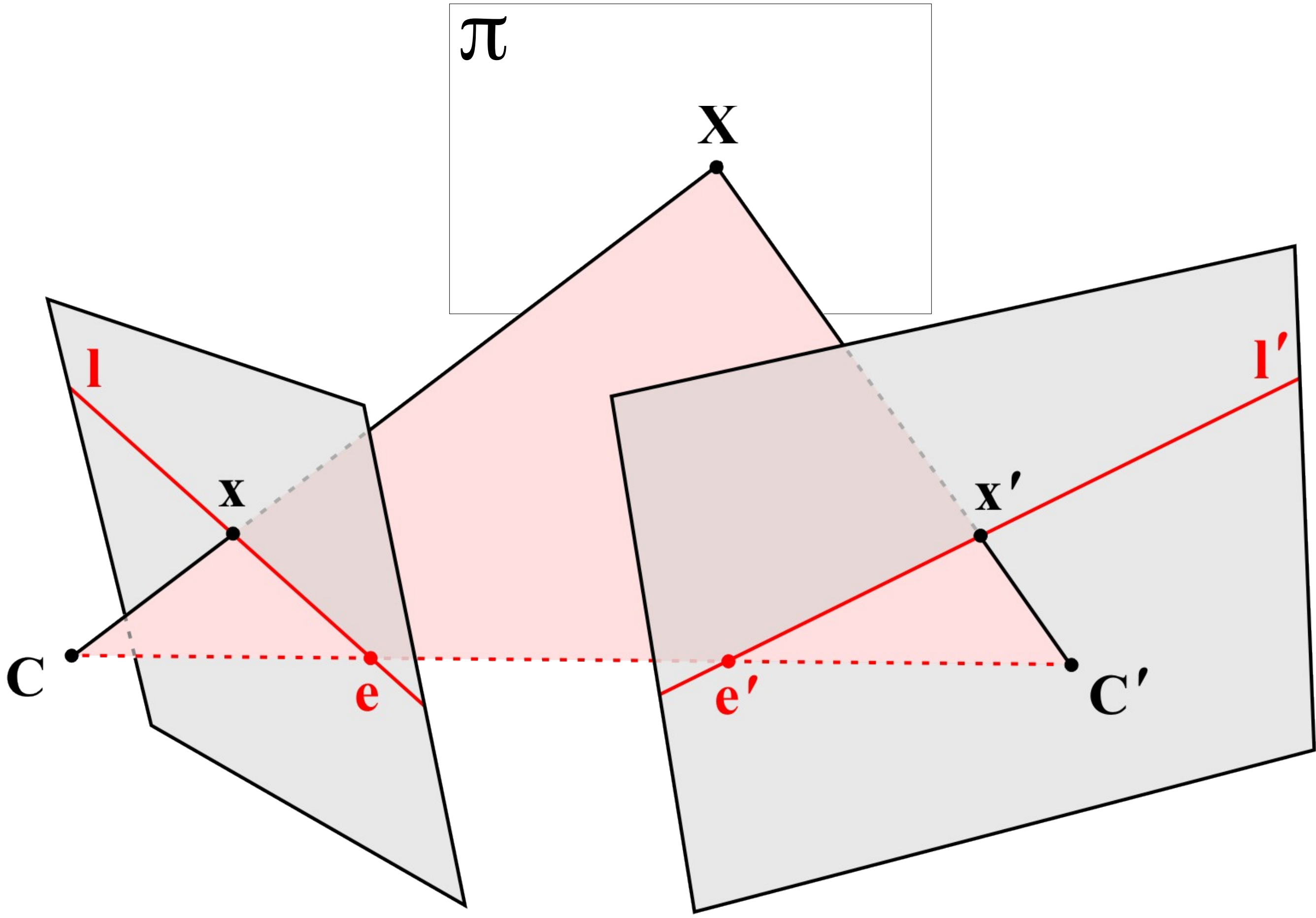


Fundamental Matrix

The fundamental matrix

$$x' = H_\pi x$$

$$\begin{aligned} l' &= e' \times x' \\ &= [e']_\times x' \\ &= [e']_\times H_\pi x \end{aligned}$$

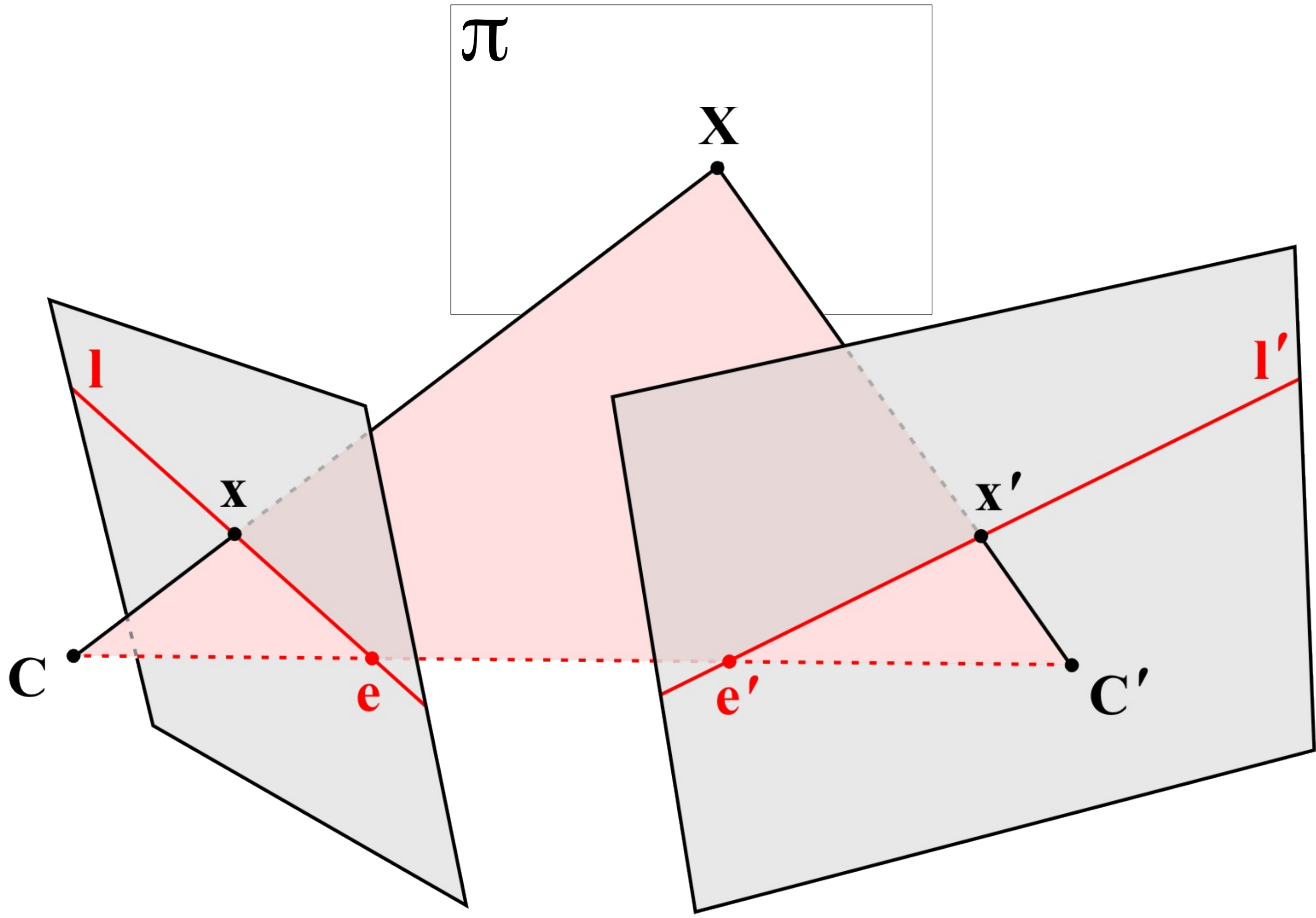


Fundamental Matrix

The fundamental matrix

$$x' = H_\pi x$$

$$\begin{aligned} l' &= e' \times x' \\ &= [e']_\times x' \\ &= [e']_\times H_\pi x \\ &= Fx \end{aligned}$$



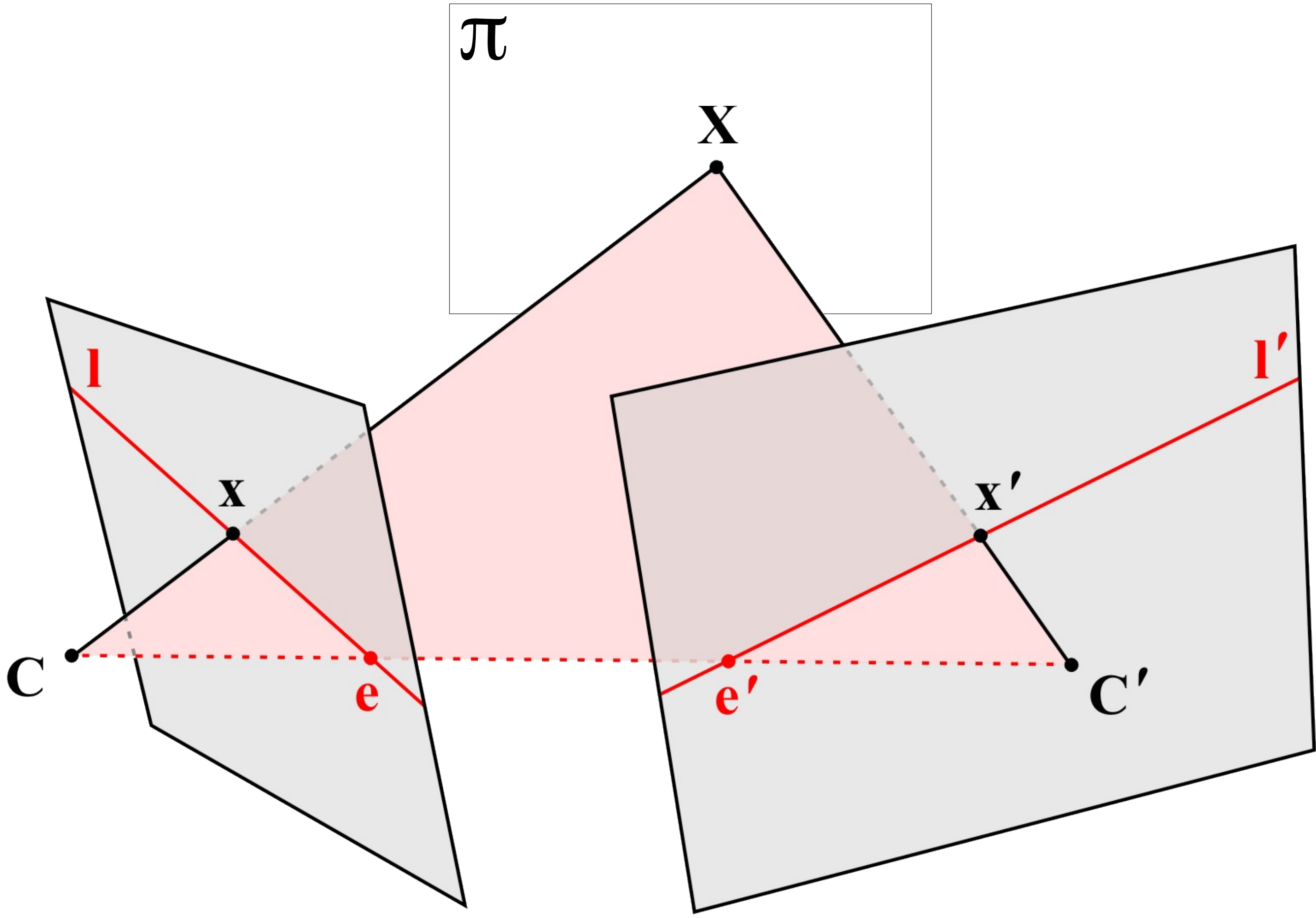
Fundamental Matrix

The fundamental matrix

$$x' = H_\pi x$$

$$\begin{aligned} l' &= e' \times x' \\ &= [e']_\times x' \\ &= [e']_\times H_\pi x \\ &= Fx \end{aligned}$$

$$0 = x'^T \cdot l'$$



Fundamental Matrix

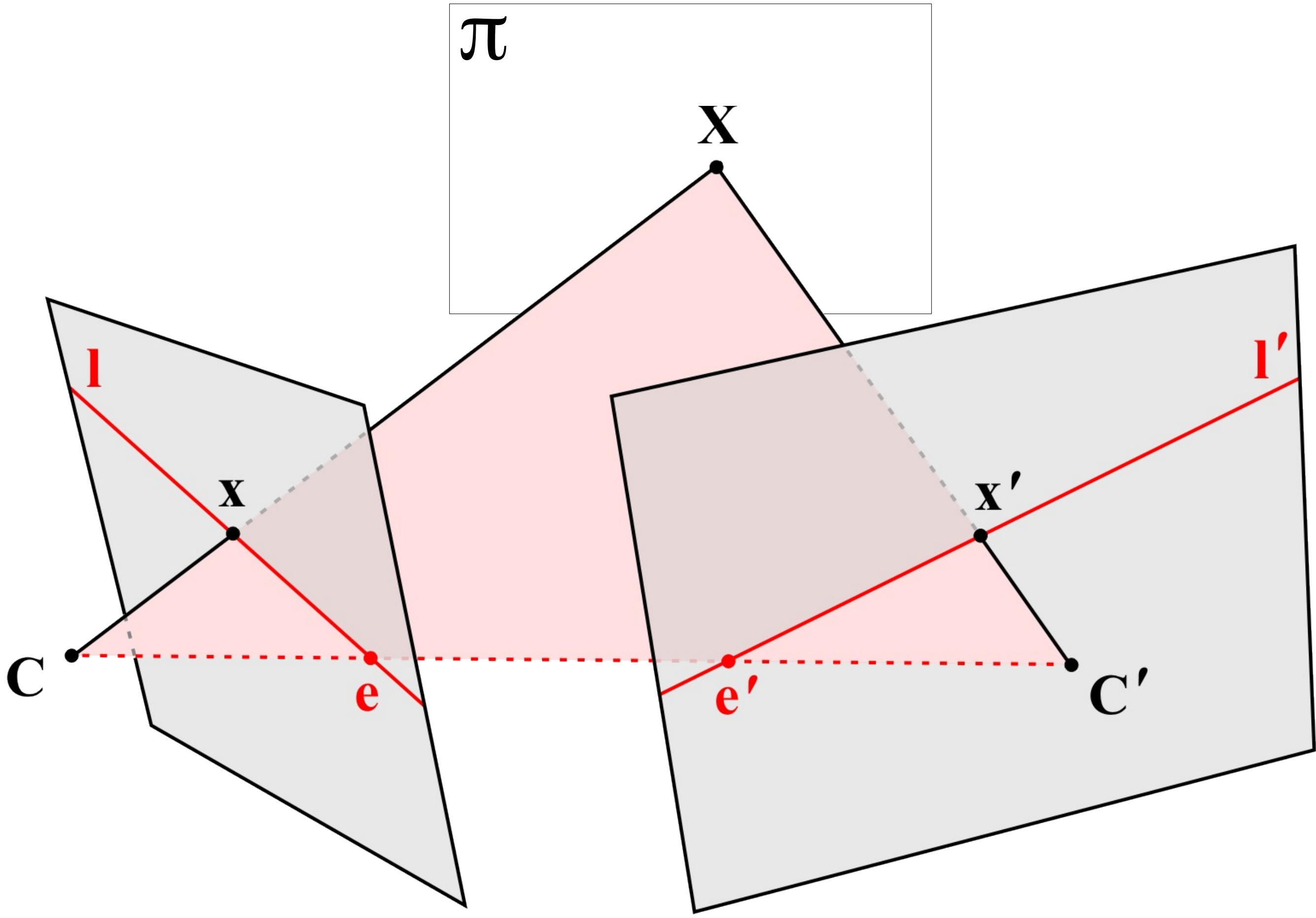
The fundamental matrix

$$x' = H_\pi x$$

$$\begin{aligned} l' &= e' \times x' \\ &= [e']_\times x' \\ &= [e']_\times H_\pi x \\ &= Fx \end{aligned}$$

$$\begin{aligned} 0 &= x'^T \cdot l' \\ &= x'^T F x \end{aligned}$$

Epipolar constraint

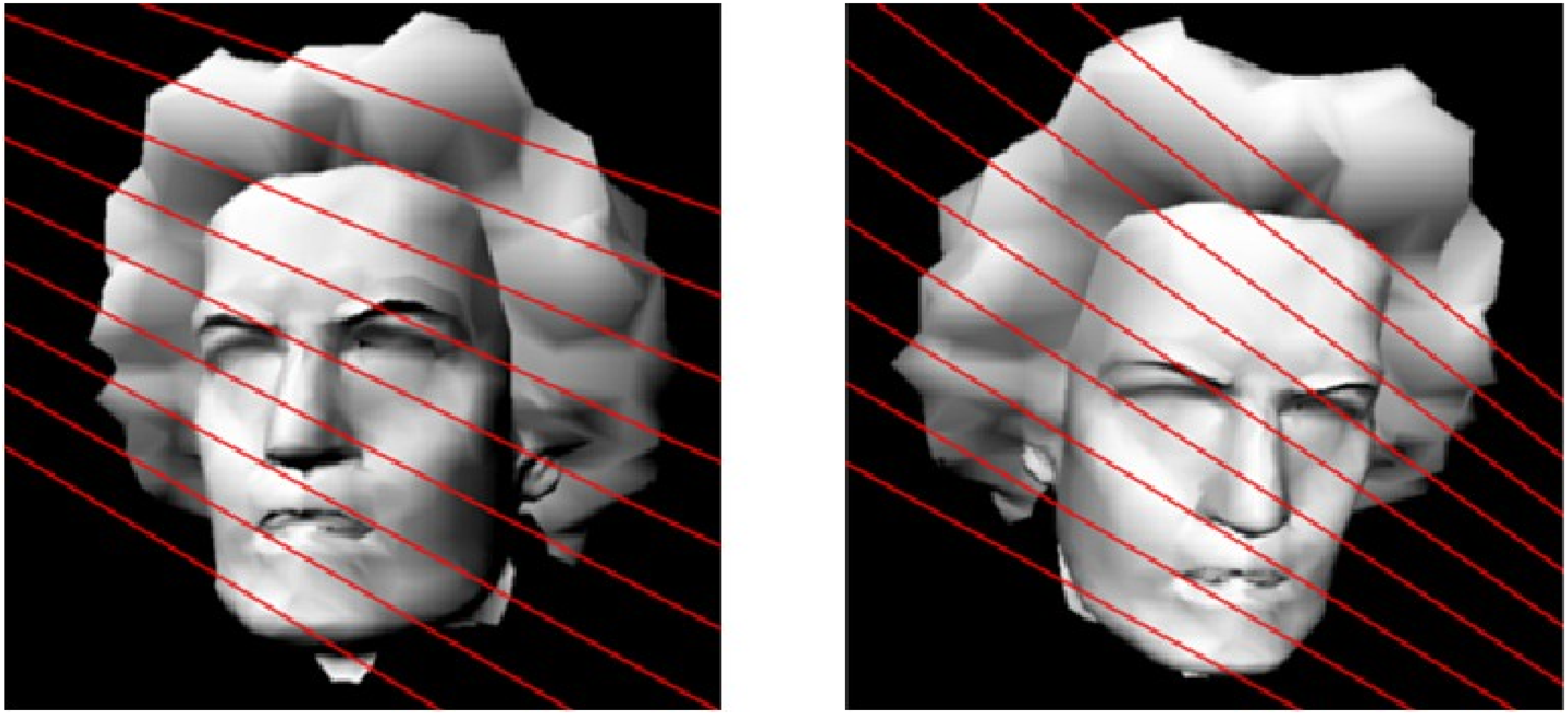


Relative Orientation

Orientation of an image pair

The relative orientation of an image pair is defined by the epipolar geometry.

Using algebraic projective geometry the epipolar geometry
can be represented by the fundamental matrix.



Fundamental Matrix Estimation

How can F be used? → Next exercise

How can F be estimated? → This exercise

Two parts:

1) Using manually clicked point pairs
   - Point pairs assumed "correct"
   - How to estimate F from point pairs?
     - Estimation very similar to previous exercises
2) Using automatically determined point pairs
   - How to get point pairs automatically?
   - How to deal with outliers?

4. Exercise

Part 1

F from (Manual) Point Pairs

4. Exercise

1. Image Acquisition:

Take a picture of a spatially structured object from two different views. Use thereby a general convergent image arrangement.

4. Exercise

1. Image Acquisition:

Take a picture of a spatially structured object from two different views.
Use thereby a general convergent image arrangement.

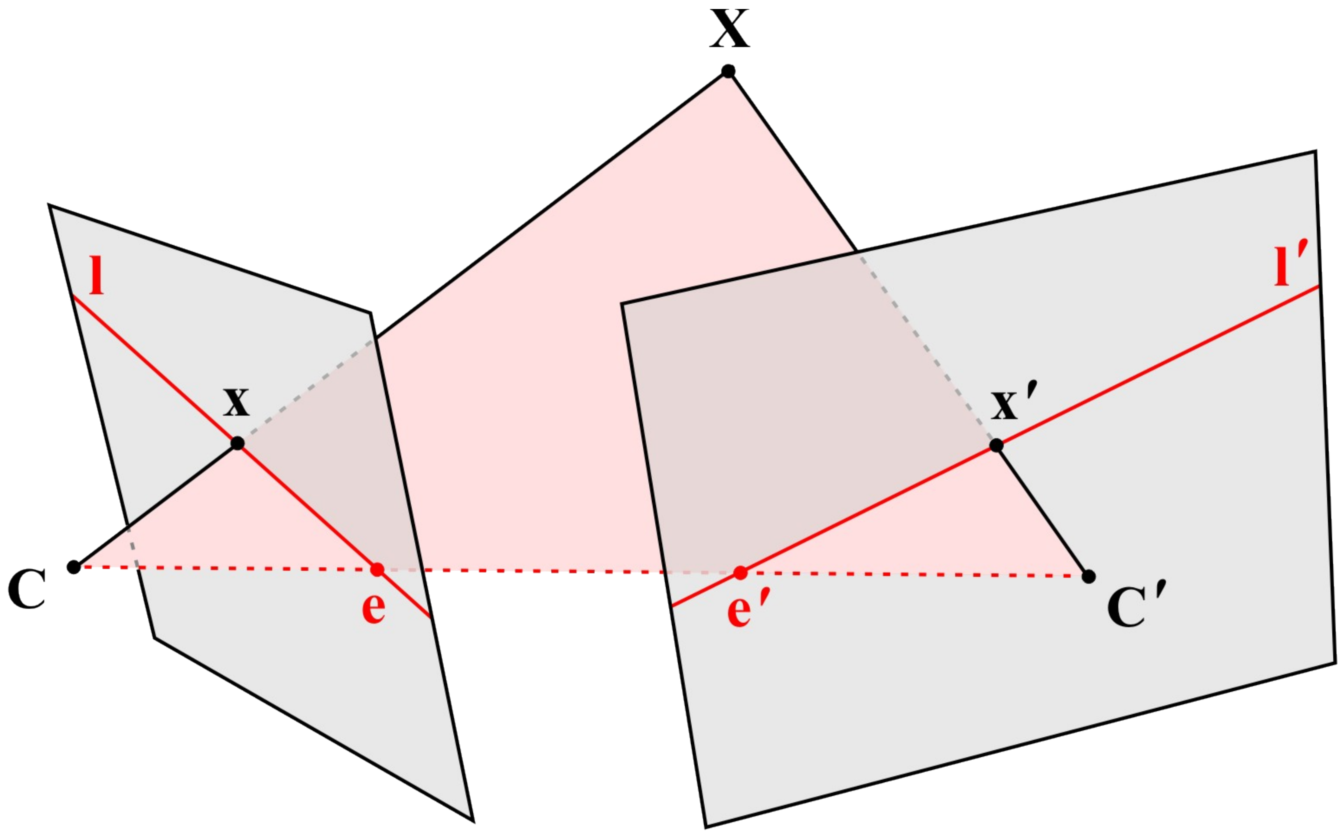
2. Homologous points:

Measure manually at least 8 homologous points $x \leftrightarrow x'$ in the image pair.

4. Exercise

3. Computation of the fundamental matrix:

Implement a function in C++ for the linear computation of the fundamental matrix F (normalized 8-point-algorithm).



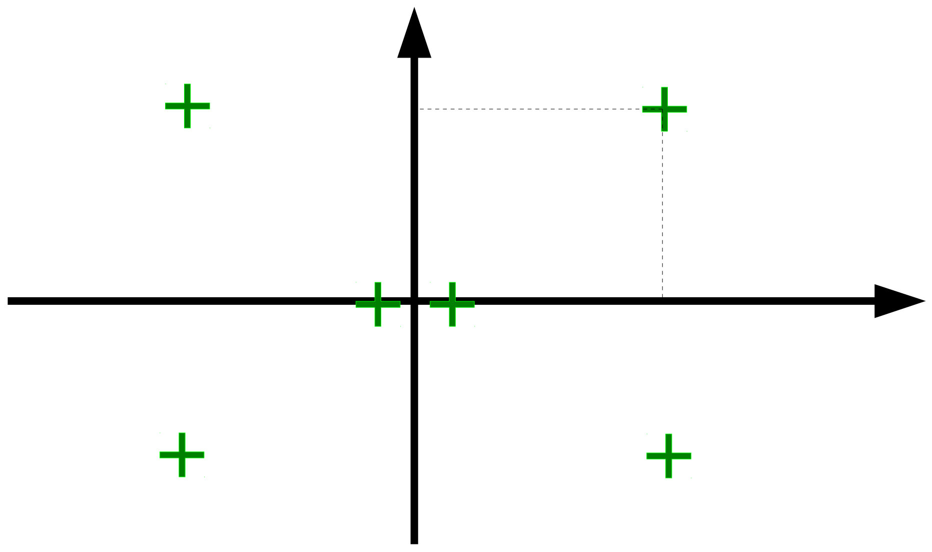
$$l' = F \cdot x$$
$$l = F^T \cdot x'$$

4. Exercise

3. Computation of the fundamental matrix:

Implement a function in C++ for the linear computation of the fundamental matrix F (normalized 8-point-algorithm).

1. Conditioning



$$T=\begin{bmatrix} \frac{1}{s_x} & 0 & \frac{-t_x}{s_x} \\ 0 & \frac{1}{s_y} & \frac{-t_y}{s_y} \\ 0 & 0 & 1 \end{bmatrix}$$

4. Exercise

3. Computation of the fundamental matrix:

Implement a function in C++ for the linear computation of the fundamental matrix F (normalized 8-point-algorithm).

2. Create design matrix

$$0 = \hat{\boldsymbol{x}}'^T_i \boldsymbol{F} \hat{\boldsymbol{x}}_i$$

4. Exercise

3. Computation of the fundamental matrix:

Implement a function in C++ for the linear computation of the fundamental matrix F (normalized 8-point-algorithm).

2. Create design matrix

$$0 = \hat{\boldsymbol{x}}'^{T}_{i} \boldsymbol{F} \hat{\boldsymbol{x}}_i$$

$$0 = \begin{bmatrix} x_i' & y_i' & w_i' \end{bmatrix} \begin{bmatrix} f_{1,1} & f_{1,2} & f_{1,3} \\ f_{2,1} & f_{2,2} & f_{2,3} \\ f_{3,1} & f_{3,2} & f_{3,3} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ w_i \end{bmatrix}$$

4. Exercise

3. Computation of the fundamental matrix:

Implement a function in C++ for the linear computation of the fundamental matrix F (normalized 8-point-algorithm).

2. Create design matrix

$$0 = \hat{\boldsymbol{x}}_i'^T \boldsymbol{F} \hat{\boldsymbol{x}}_i$$

$$0 = \begin{bmatrix} x_i' & y_i' & w_i' \end{bmatrix} \begin{bmatrix} f_{1,1} & f_{1,2} & f_{1,3} \\ f_{2,1} & f_{2,2} & f_{2,3} \\ f_{3,1} & f_{3,2} & f_{3,3} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ w_i \end{bmatrix}$$

$$0 = \begin{bmatrix} x_i' & y_i' & w_i' \end{bmatrix} \begin{bmatrix} f_{1,1} x_i + f_{1,2} y_i + f_{1,3} w_i \\ f_{2,1} x_i + f_{2,2} y_i + f_{2,3} w_i \\ f_{3,1} x_i + f_{3,2} y_i + f_{3,3} w_i \end{bmatrix}$$

4. Exercise

3. Computation of the fundamental matrix:

Implement a function in C++ for the linear computation of the fundamental matrix F (normalized 8-point-algorithm).

2. Create design matrix

$$0 = \hat{\boldsymbol{x}}'^{T}_{i} \boldsymbol{F} \hat{\boldsymbol{x}}_i$$

$$0 = \begin{bmatrix} x'_i & y'_i & w'_i \end{bmatrix} \begin{bmatrix} f_{1,1} & f_{1,2} & f_{1,3} \\ f_{2,1} & f_{2,2} & f_{2,3} \\ f_{3,1} & f_{3,2} & f_{3,3} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ w_i \end{bmatrix}$$

$$\begin{aligned} 0 = {} & f_{1,1} x_i x'_i + f_{1,2} y_i x'_i + f_{1,3} w_i x'_i \\ & + f_{2,1} x_i y'_i + f_{2,2} y_i y'_i + f_{2,3} w_i y'_i \\ & + f_{3,1} x_i w'_i + f_{3,2} y_i w'_i + f_{3,3} w_i w'_i \end{aligned}$$

4. Exercise

3. Computation of the fundamental matrix:

Implement a function in C++ for the linear computation of the fundamental matrix F (normalized 8-point-algorithm).

2. Create design matrix

$$0 = \hat{\boldsymbol{x}}'^T_i \boldsymbol{F} \hat{\boldsymbol{x}}_i$$

$$0 = \begin{bmatrix} x_i' & y_i' & w_i' \end{bmatrix} \begin{bmatrix} f_{1,1} & f_{1,2} & f_{1,3} \\ f_{2,1} & f_{2,2} & f_{2,3} \\ f_{3,1} & f_{3,2} & f_{3,3} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ w_i \end{bmatrix}$$

$$0 = \begin{bmatrix} x_i x_i' & y_i x_i' & w_i x_i' & x_i y_i' & y_i y_i' & w_i y_i' & x_i w_i' & y_i w_i' & w_i w_i' \end{bmatrix} \begin{bmatrix} f_{1,1} \\ f_{1,2} \\ f_{1,3} \\ \vdots \\ f_{3,3} \end{bmatrix}$$

4. Exercise

3. Computation of the fundamental matrix:

Implement a function in C++ for the linear computation of the fundamental matrix F (normalized 8-point-algorithm).

2. Create design matrix

$$A = \begin{bmatrix} \hat{x}_1 \hat{x}'_1 & \hat{y}_1 \hat{x}'_1 & \hat{w}_1 \hat{x}'_1 & \hat{x}_1 \hat{y}'_1 & \hat{y}_1 \hat{y}'_1 & \hat{w}_1 \hat{y}'_1 & \hat{x}_1 \hat{w}'_1 & \hat{y}_1 \hat{w}'_1 & \hat{w}_1 \hat{w}'_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \hat{x}_n \hat{x}'_n & \hat{y}_n \hat{x}'_n & \hat{w}_n \hat{x}'_n & \hat{x}_n \hat{y}'_n & \hat{y}_n \hat{y}'_n & \hat{w}_n \hat{y}'_n & \hat{x}_n \hat{w}'_n & \hat{y}_n \hat{w}'_n & \hat{w}_n \hat{w}'_n \end{bmatrix}$$

4. Exercise

3. Computation of the fundamental matrix:

Implement a function in C++ for the linear computation of the fundamental matrix F (normalized 8-point-algorithm).

2. Create design matrix

$$A = \begin{bmatrix} \hat{x}_1 \hat{x}'_1 & \hat{y}_1 \hat{x}'_1 & \hat{w}_1 \hat{x}'_1 & \hat{x}_1 \hat{y}'_1 & \hat{y}_1 \hat{y}'_1 & \hat{w}_1 \hat{y}'_1 & \hat{x}_1 \hat{w}'_1 & \hat{y}_1 \hat{w}'_1 & \hat{w}_1 \hat{w}'_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \hat{x}_n \hat{x}'_n & \hat{y}_n \hat{x}'_n & \hat{w}_n \hat{x}'_n & \hat{x}_n \hat{y}'_n & \hat{y}_n \hat{y}'_n & \hat{w}_n \hat{y}'_n & \hat{x}_n \hat{w}'_n & \hat{y}_n \hat{w}'_n & \hat{w}_n \hat{w}'_n \end{bmatrix}$$

3. Solve equation system with SVD and reshape

$$\boldsymbol{Af} = \boldsymbol{0} \qquad \tilde{\boldsymbol{F}} = \begin{bmatrix} f_1 & f_2 & f_3 \\ f_4 & f_5 & f_6 \\ f_7 & f_8 & f_9 \end{bmatrix}$$

4. Exercise

3. Computation of the fundamental matrix:

Implement a function in C++ for the linear computation of the fundamental matrix F (normalized 8-point-algorithm).

4. Enforce singularity

Wanted:
$$\boxed{\begin{array}{l} \left\| \boldsymbol{F} - \boldsymbol{F}^* \right\| = min \\ \left| \boldsymbol{F}^* \right| = 0 \end{array}}$$

4. Exercise

3. Computation of the fundamental matrix:

Implement a function in C++ for the linear computation of the fundamental matrix F (normalized 8-point-algorithm).

4. Enforce singularity

Wanted:
$$\boxed{\begin{aligned} &\left\|\boldsymbol{F}-\boldsymbol{F}^*\right\| = min \\ &\left|\boldsymbol{F}^*\right| = 0 \end{aligned}}$$

$$\xrightarrow{\text{SVD}}$$

$$\tilde{\boldsymbol{F}} = \tilde{\boldsymbol{U}}\, diag(d_1, d_2, d_3)\, \tilde{\boldsymbol{V}}^T$$

4. Exercise

3. Computation of the fundamental matrix:

Implement a function in C++ for the linear computation of the fundamental matrix F (normalized 8-point-algorithm).

4. Enforce singularity

Wanted: $\boxed{\begin{array}{l}\left\|\boldsymbol{F}-\boldsymbol{F}^*\right\|=min \\ \left|\boldsymbol{F}^*\right|=0\end{array}}$ $\xrightarrow{\text{SVD}}$ $\begin{array}{l}\tilde{\boldsymbol{F}}=\tilde{\boldsymbol{U}}\, diag(d_1, d_2, d_3)\,\tilde{\boldsymbol{V}}^T \\ \tilde{\boldsymbol{F}}^*=\tilde{\boldsymbol{U}}\, diag(d_1, d_2, 0)\,\tilde{\boldsymbol{V}}^T\end{array}$

4. Exercise

3. Computation of the fundamental matrix:

Implement a function in C++ for the linear computation of the fundamental matrix F (normalized 8-point-algorithm).

4. Enforce singularity

Wanted:
$$\boxed{\begin{array}{l} \|\boldsymbol{F}-\boldsymbol{F}^*\| = min \\ |\boldsymbol{F}^*| = 0 \end{array}} \xrightarrow{\text{SVD}} \begin{array}{l} \tilde{\boldsymbol{F}} = \tilde{\boldsymbol{U}}\, diag(d_1, d_2, d_3)\, \tilde{\boldsymbol{V}}^T \\ \tilde{\boldsymbol{F}}^* = \tilde{\boldsymbol{U}}\, diag(d_1, d_2, 0)\, \tilde{\boldsymbol{V}}^T \end{array}$$

5. Deconditioning

$$\boldsymbol{F} = \boldsymbol{T}'^T \tilde{\boldsymbol{F}}^* \boldsymbol{T}$$

4. Exercise

4. Visualization of the fundamental matrix by epipolar lines:

a) Mark the used points in both images
b) and draw the associated epipolar lines in the corresponding image.

4. Exercise

4. Visualization of the fundamental matrix by epipolar lines:

a) Mark the used points in both images
b) and draw the associated epipolar lines in the corresponding image.

5. Evaluation:

a) Show the image pair and comment the line characteristics in brief.
b) Calculate the (average) geometric image error (symmetric epipolar distance) of F for all points.

$$d=\frac{1}{N}\sum_{i=1}^{N}\frac{\left(\boldsymbol{x}_i'^{\mathrm{T}}\boldsymbol{F}\,\boldsymbol{x}_i\right)^2}{\left(\boldsymbol{F}\,\boldsymbol{x}_i\right)_a^2+\left(\boldsymbol{F}\,\boldsymbol{x}_i\right)_b^2+\left(\boldsymbol{F}^T\boldsymbol{x}_i'\right)_a^2+\left(\boldsymbol{F}^T\boldsymbol{x}_i'\right)_b^2}$$

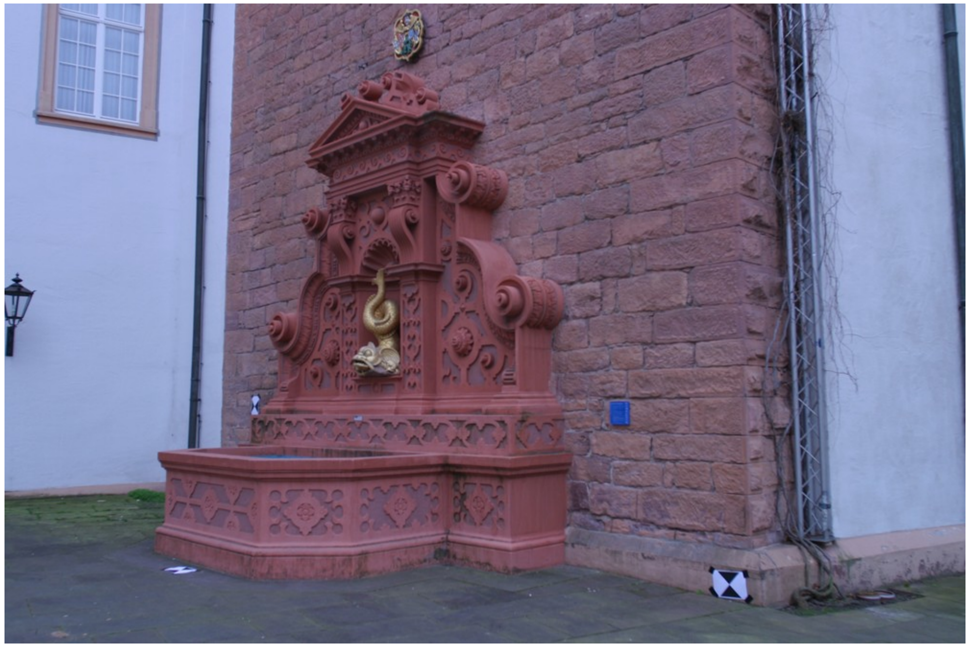
c) Count the number of inliers. Inliers are point pairs whose error is smaller than a predefined threshold (e.g. 2 pixels)

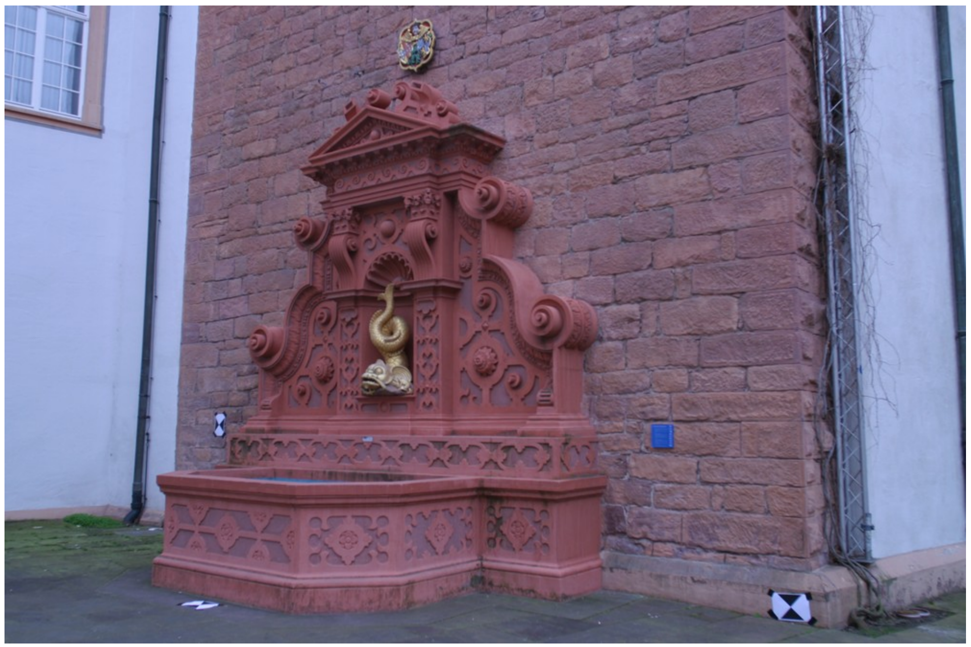
4. Exercise

Part 2

Automatic Point Pairs and Robust Estimation

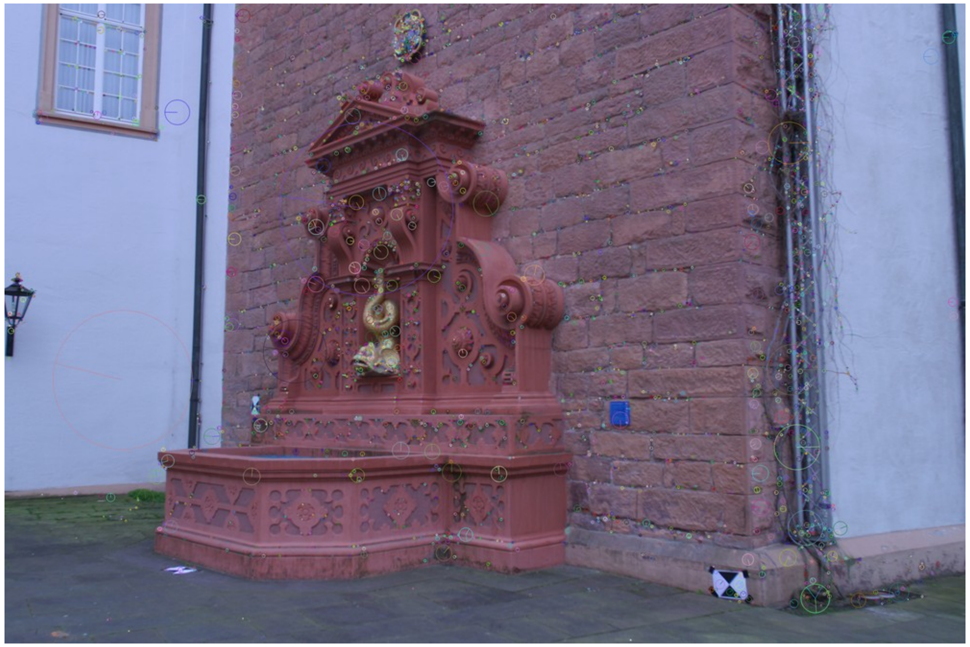
Keypoints

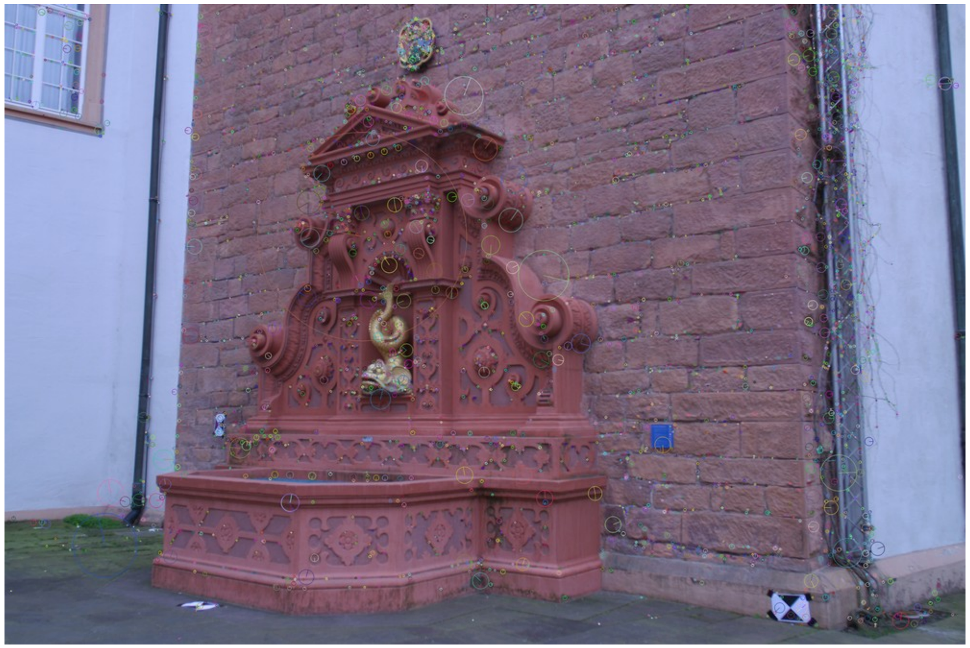




How to automatically establish points correspondences?

Keypoints





Step 1: Keypoint detection (in each image)

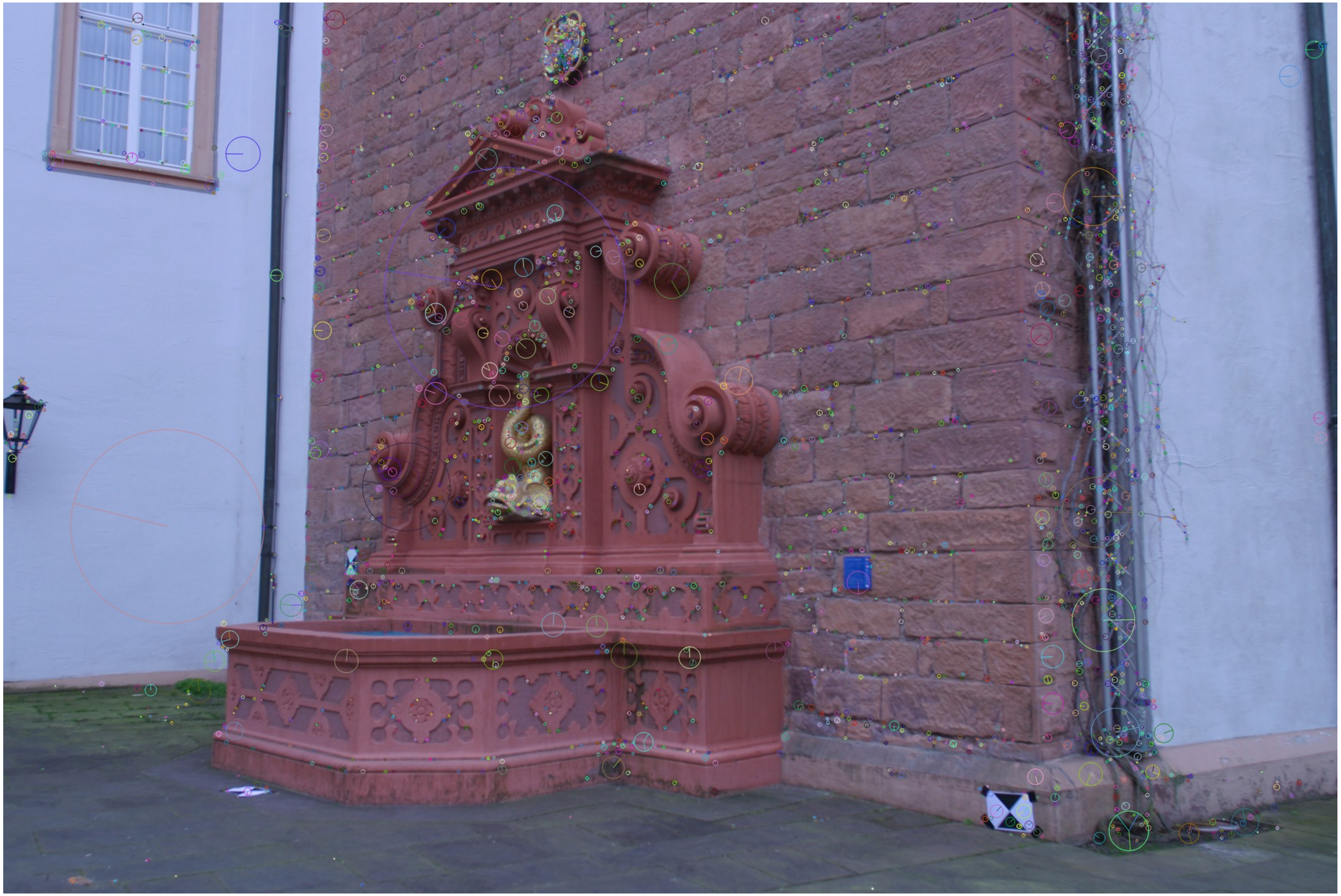
→ Find “salient” points (landmarks)

→ In a reproducible way

→ Usually:

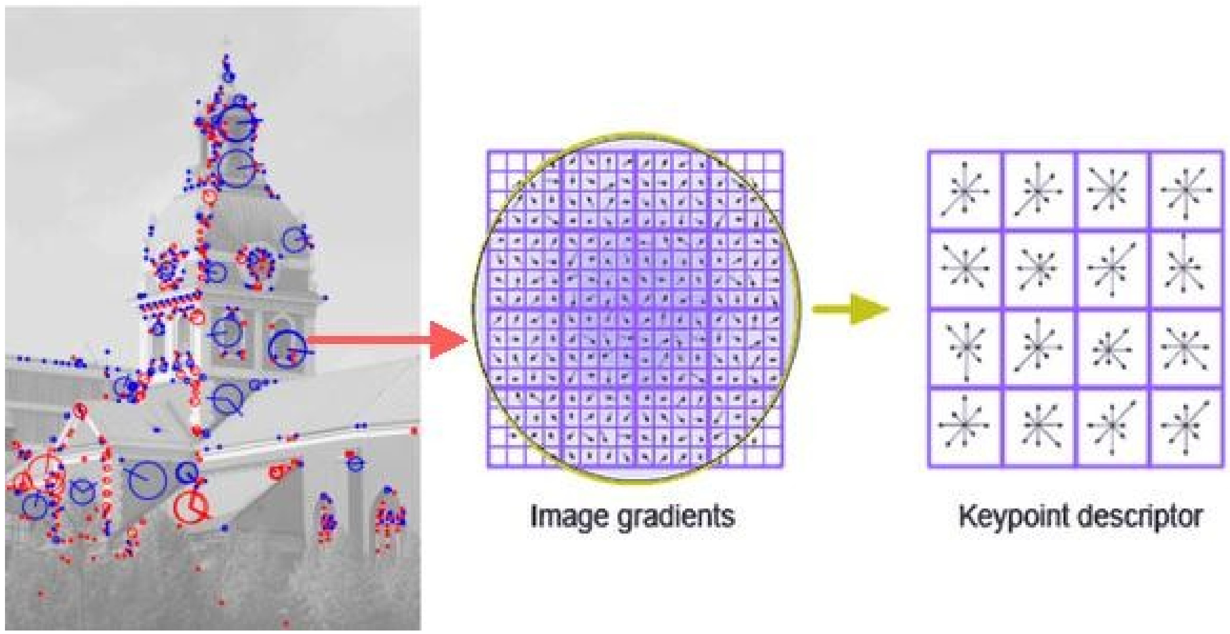
- Bright spots on dark ground (and vice versa)
- Corners

Keypoint Detection



Keypoint Description

Step 2: Keypoint Description (for each image and keypoint)
→ Use information from detector (location, scale, angle)
→ Compute “descriptor” (vector) that describes surroundings
→ Again, in a stable, reproducible way



SIFT Detection
→ Position
→ Scale
→ Orientation

SIFT Description
→ gradient histogram
→ 128 dimensional vector

Keypoint Matching

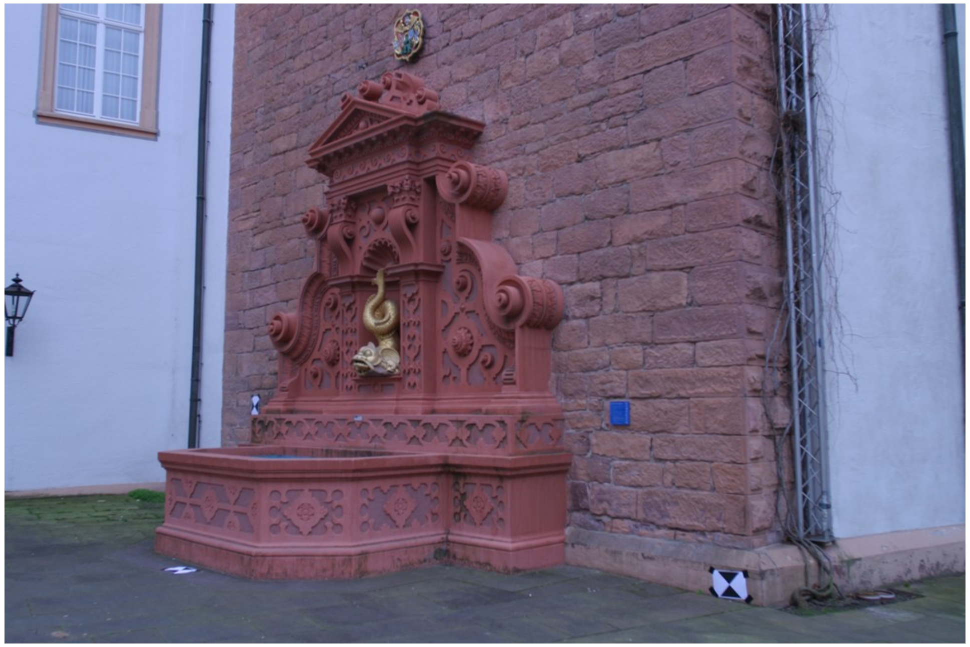
Step 3: Keypoint Matching (for each image pair)

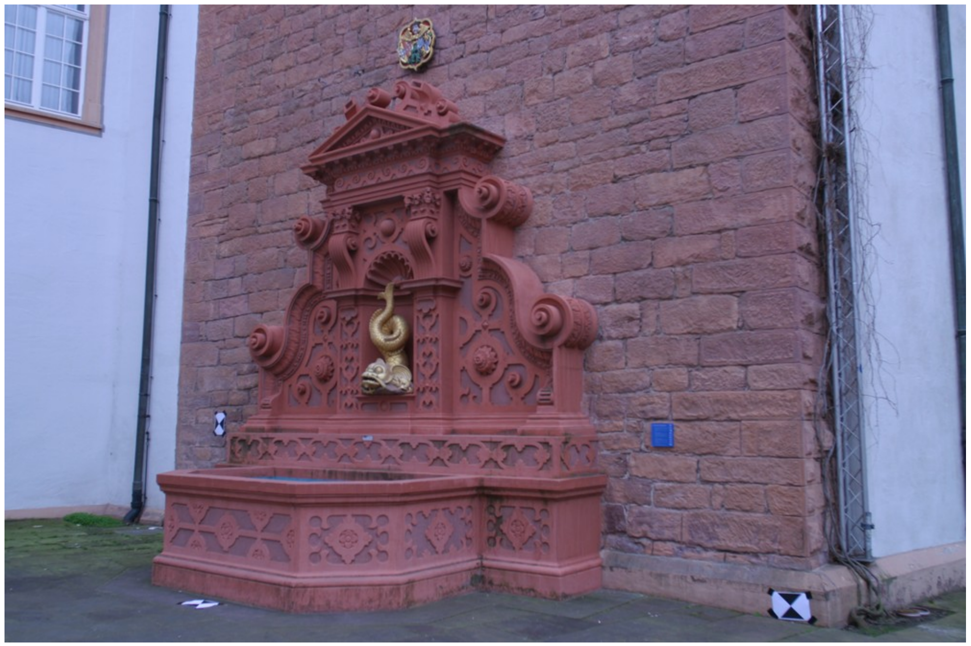
- For each keypoint (descriptor) in first image
  find most similar keypoint (descriptor) in second image

- Similarity metric depends on descriptor
  - Real valued descriptors: Euclidean distance or dot product
  - Binary descriptors: Hamming distance

- Optimizations:
  → Run in parallel
  → Data structures (trees, grids)
  → Partial distance computation

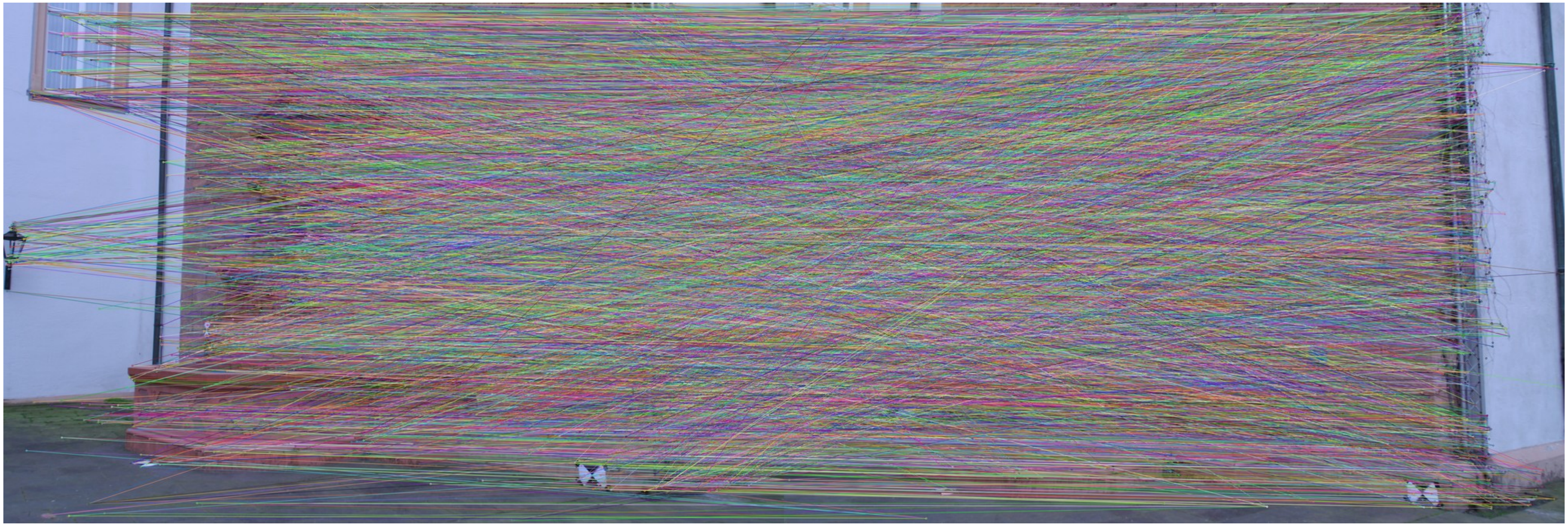
Keypoints





How to automatically establish points correspondences?

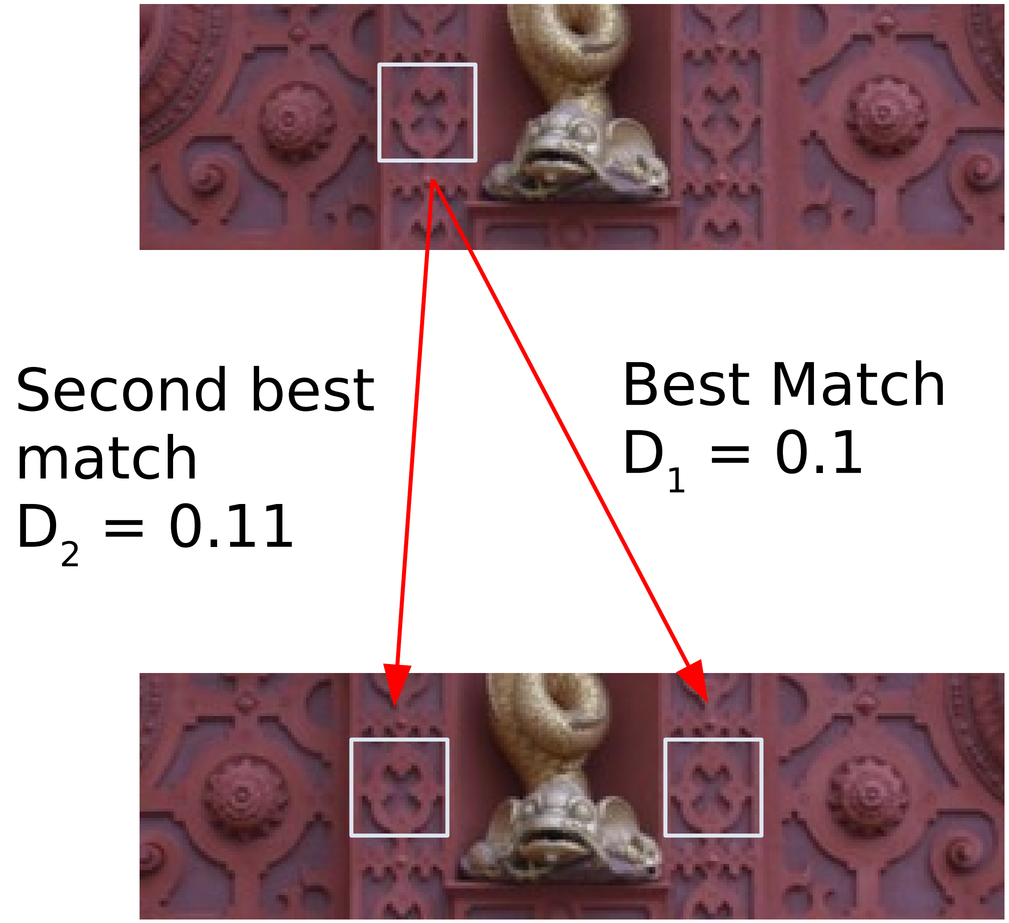
Keypoint Matching



Large amount of gross errors (outliers)!

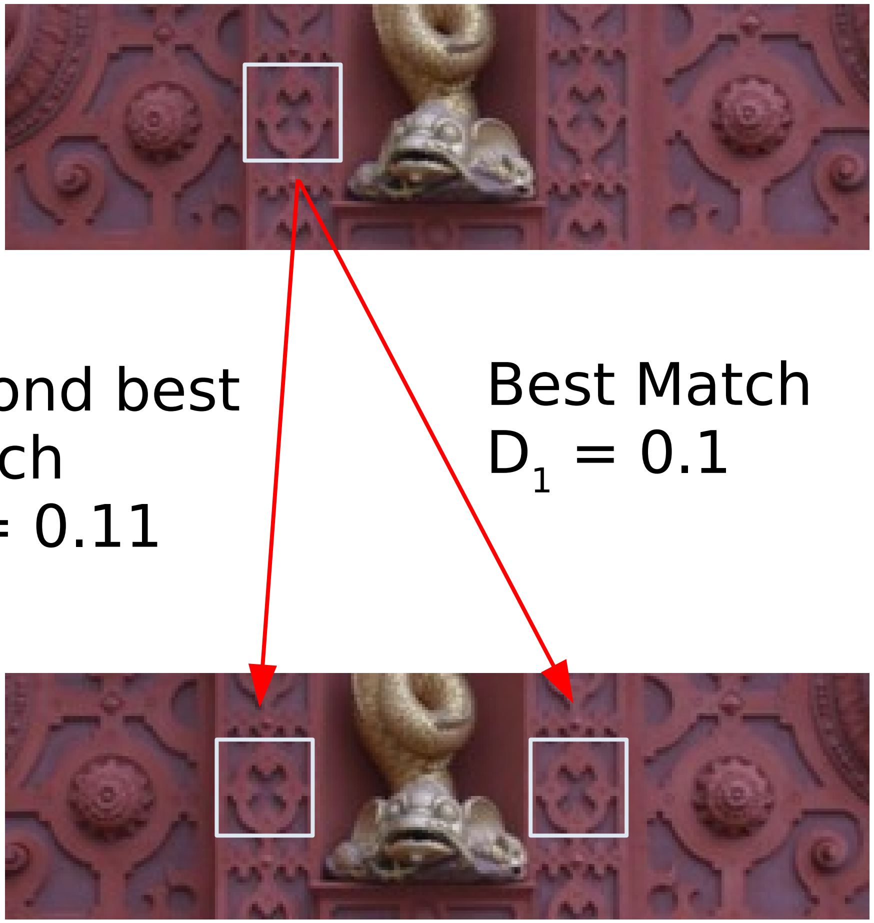
Filtering

1) Ratio Test



Filtering

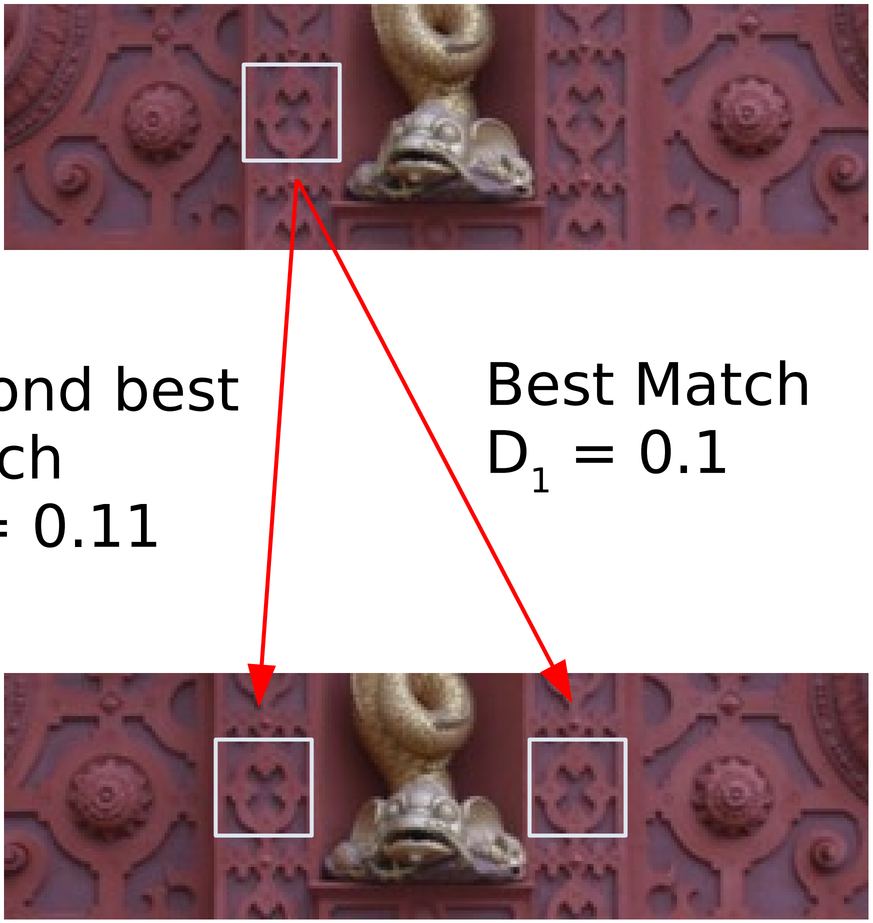
1) Ratio Test



$D_1/D_2 = 0.91 > 0.75 \rightarrow$ Reject

Filtering

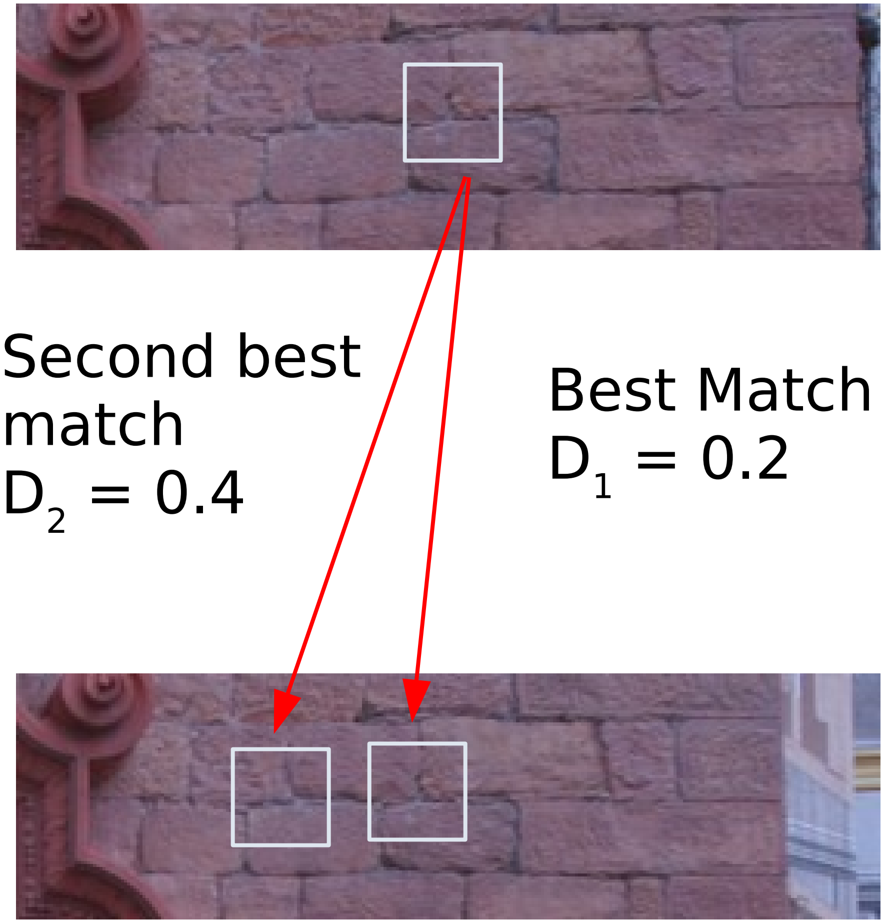
1) Ratio Test



Second best match
$D_2 = 0.11$

Best Match
$D_1 = 0.1$

$D_1/D_2 = 0.91 > 0.75 \rightarrow$ Reject



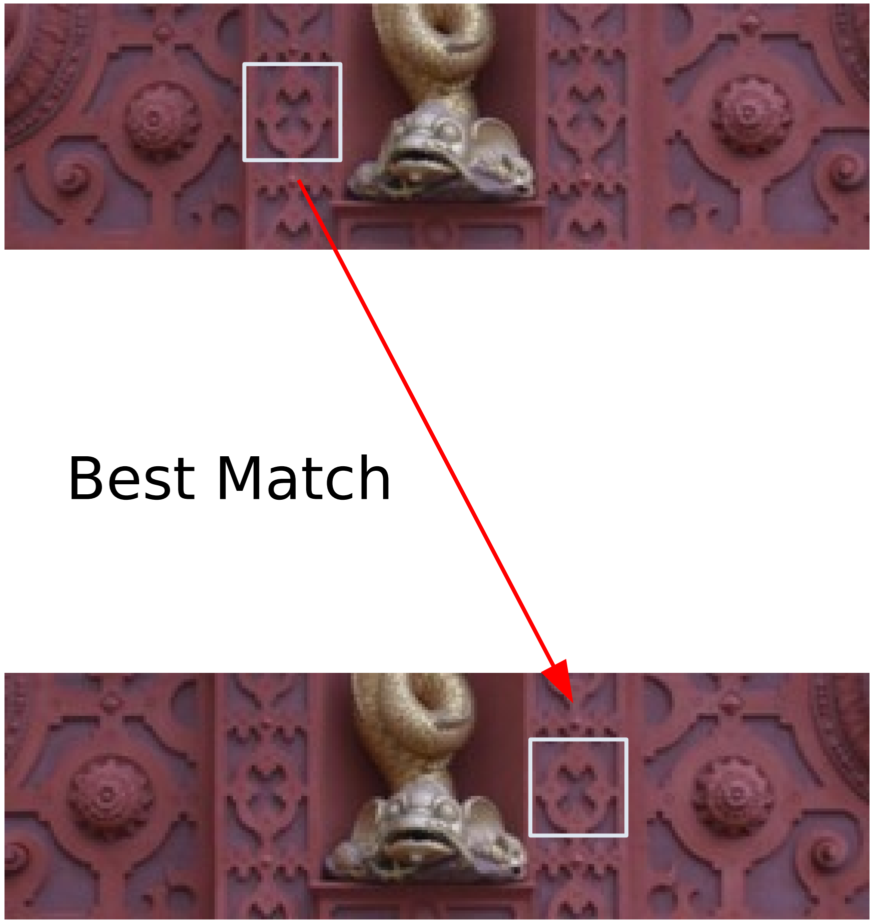
Second best match
$D_2 = 0.4$

Best Match
$D_1 = 0.2$

$D_1/D_2 = 0.5 < 0.75 \rightarrow$ Accept

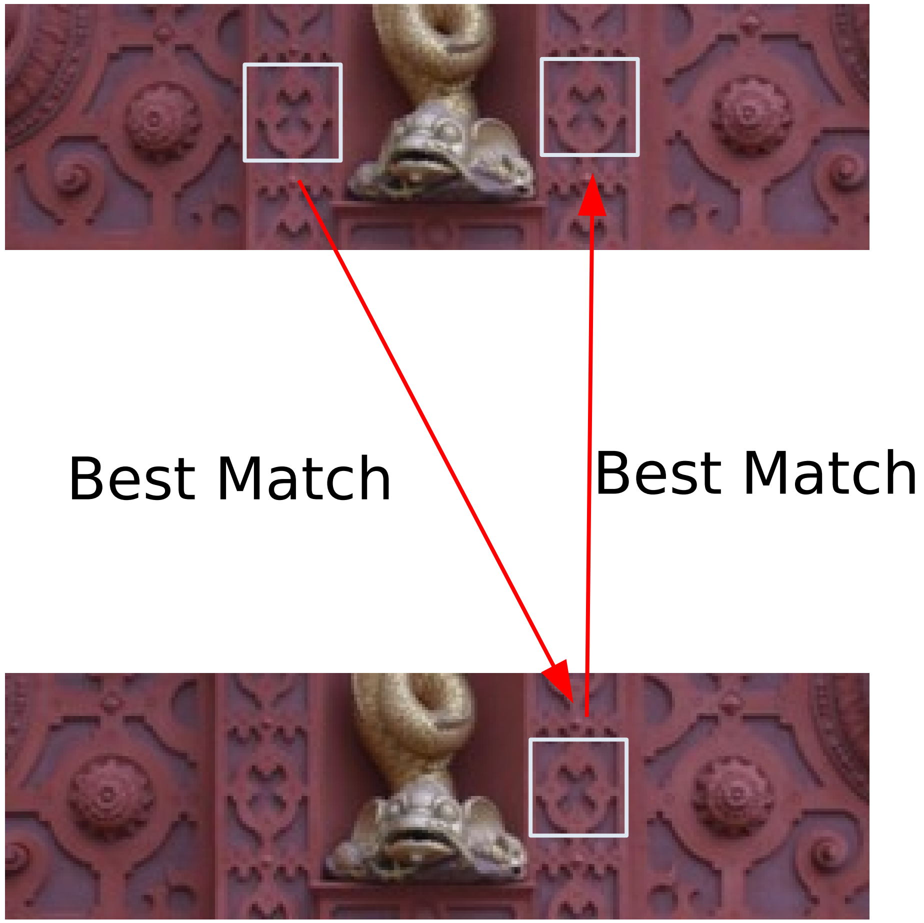
Filtering

2) Cross Consistency Check



Filtering

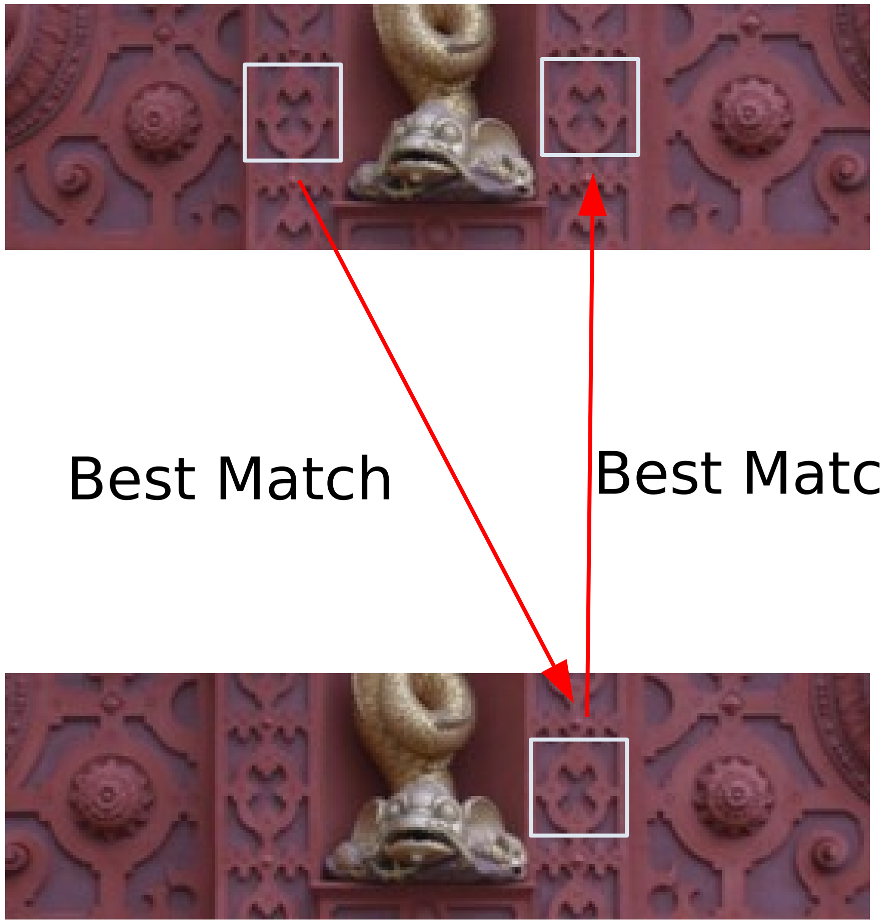
2) Cross Consistency Check



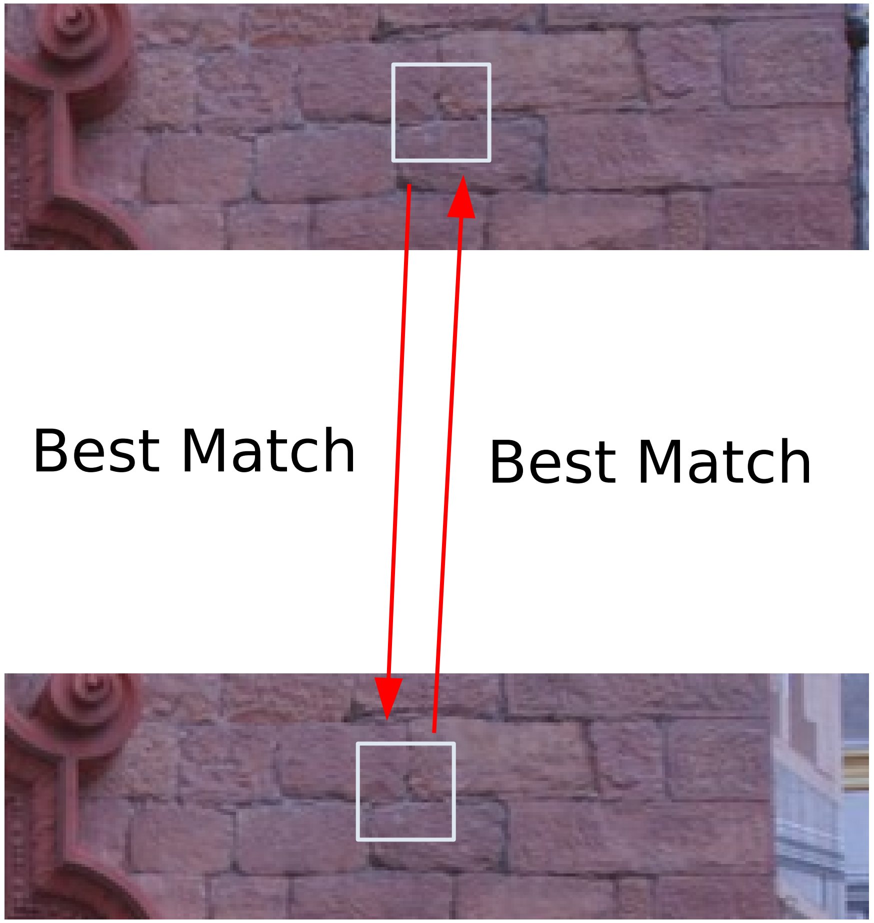
Matches inconsistent → Reject

Filtering

2) Cross Consistency Check



Matches inconsistent → Reject



Matches consistent → Accept

RANSAC

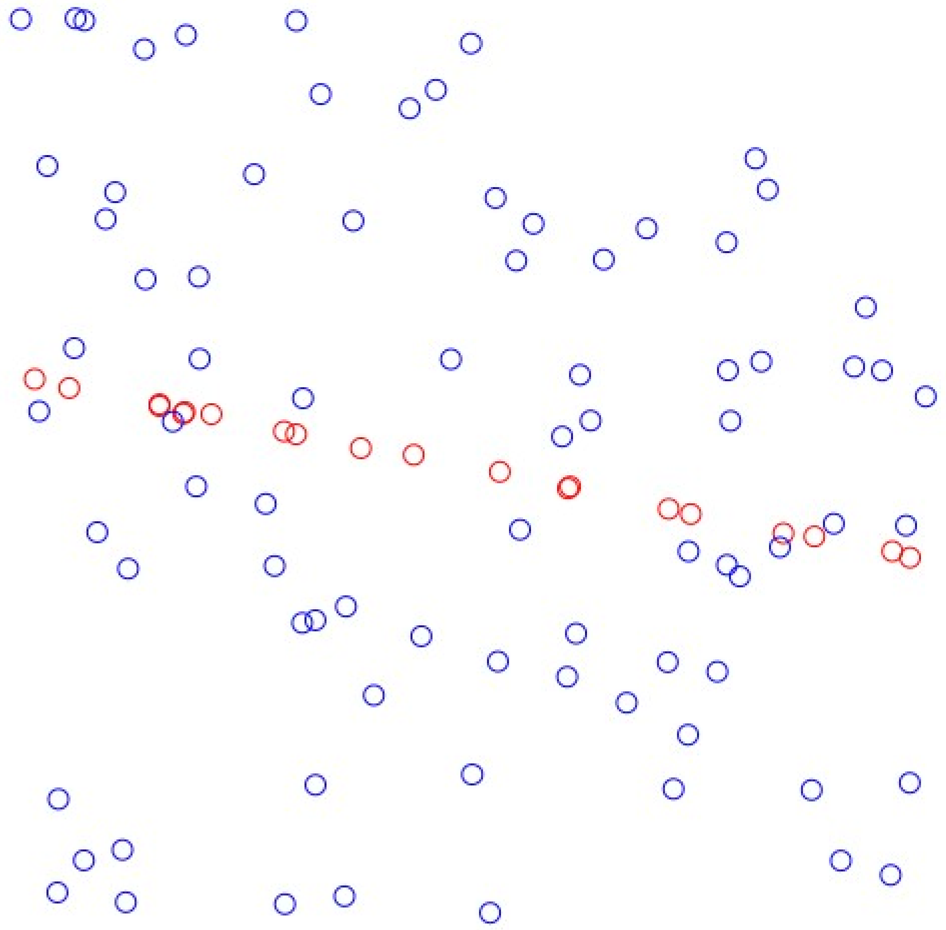
RANdom Sampling Consensus

Requirements:
- A model to be estimated from few data points
- Lots more data points than necessary

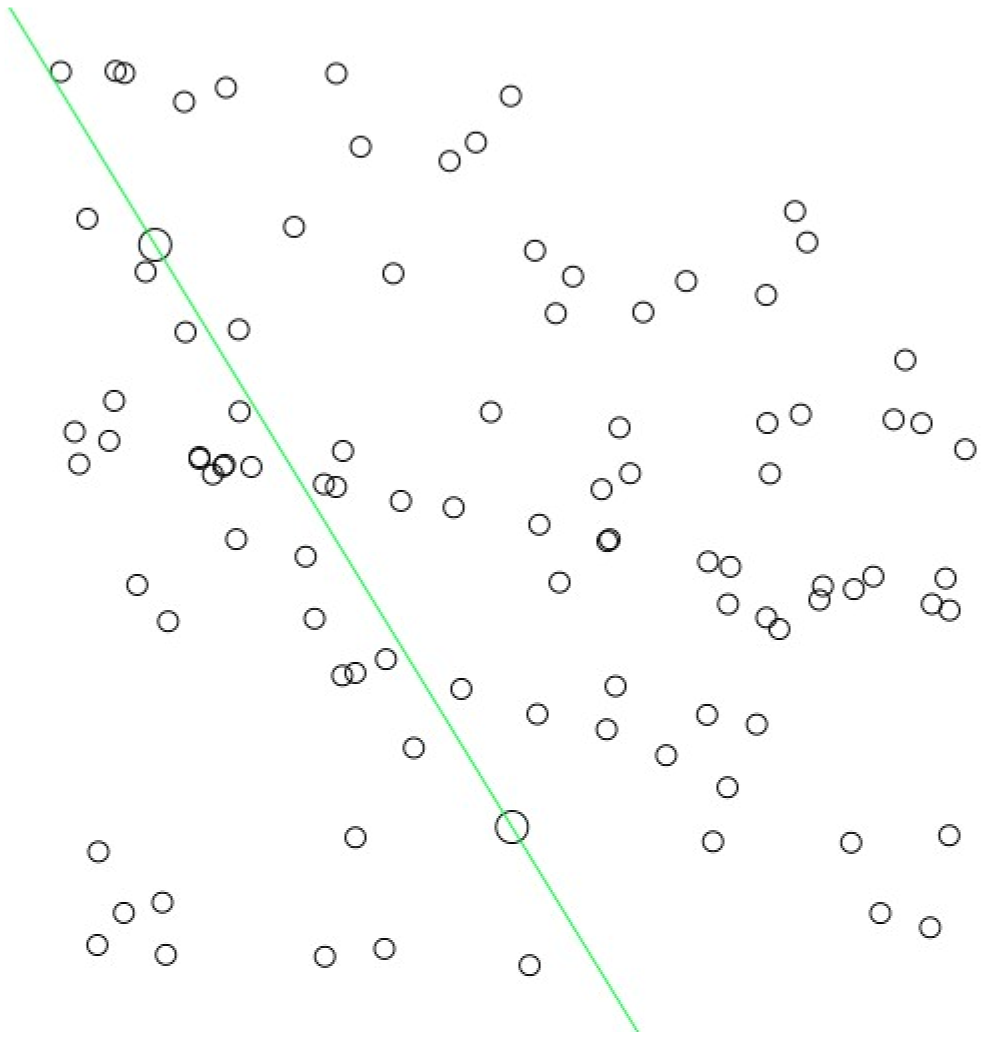
Steps:
1) Select random subset
2) Estimate model from subset
3) Evaluate model on all data (e.g. count inliers)
4) Repeat couple thousand times and keep the best model

Key idea:
1) Outliers don’t agree on a model!
2) Sooner or later, the small, random subset will be all inliers

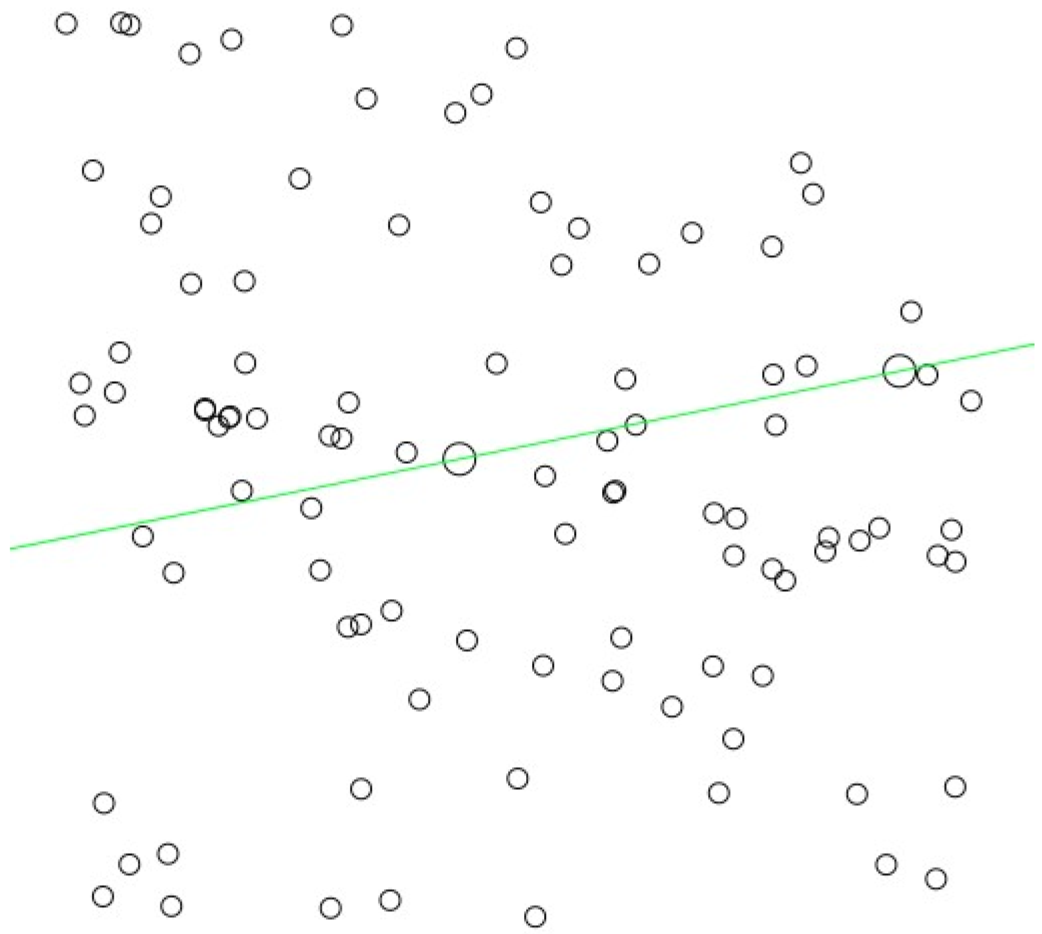
RANSAC



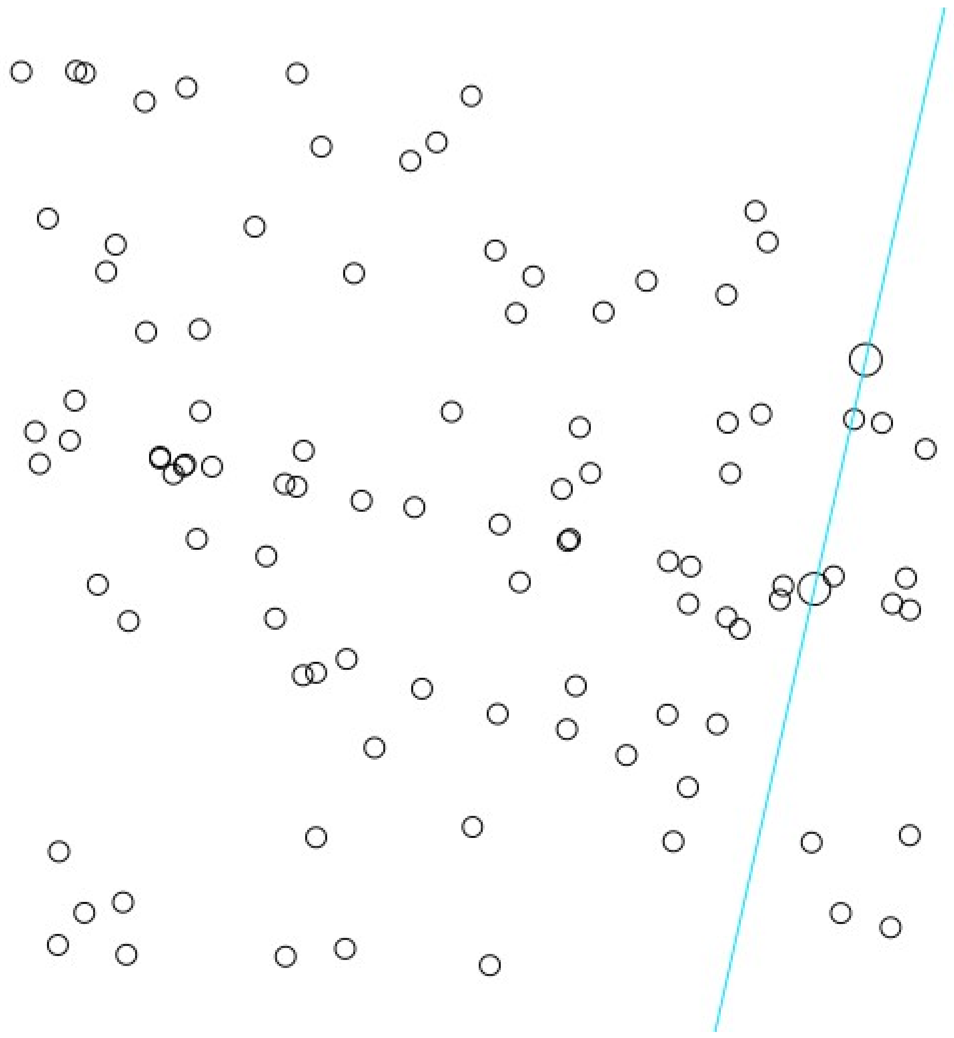
RANSAC



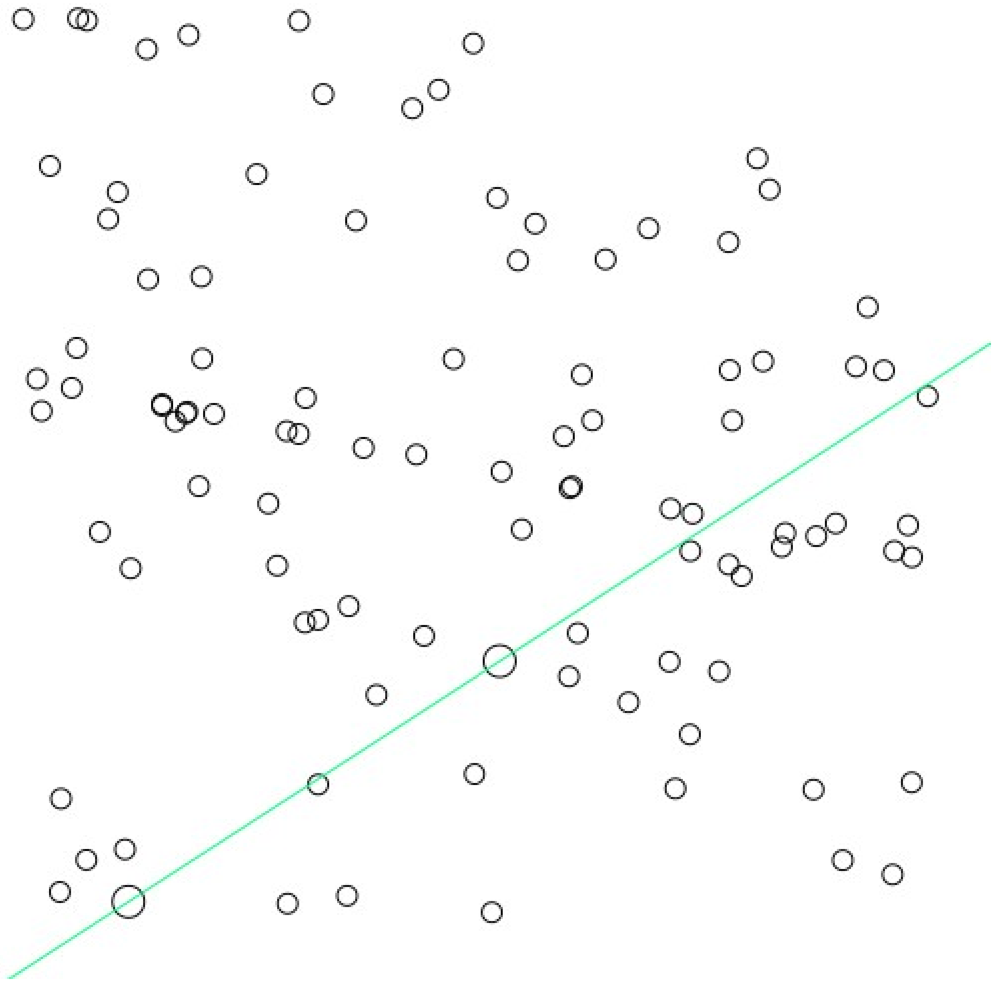
RANSAC



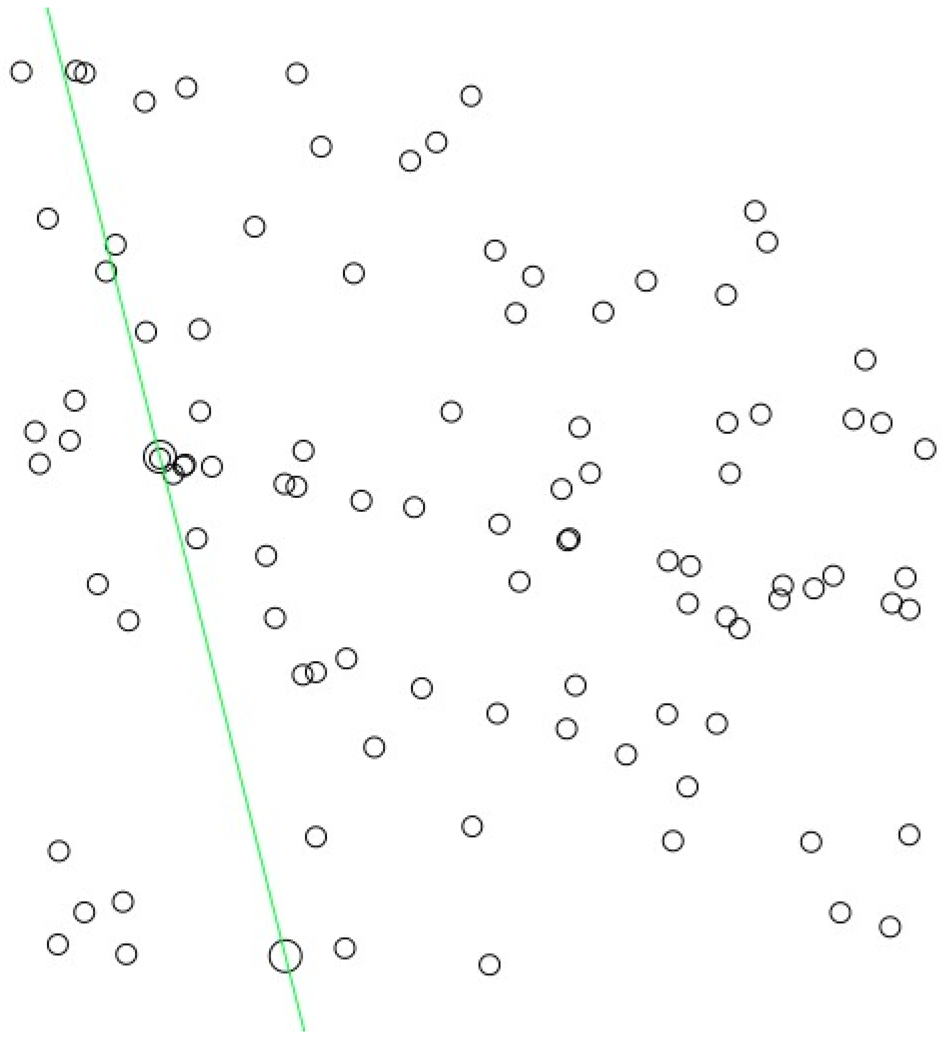
RANSAC



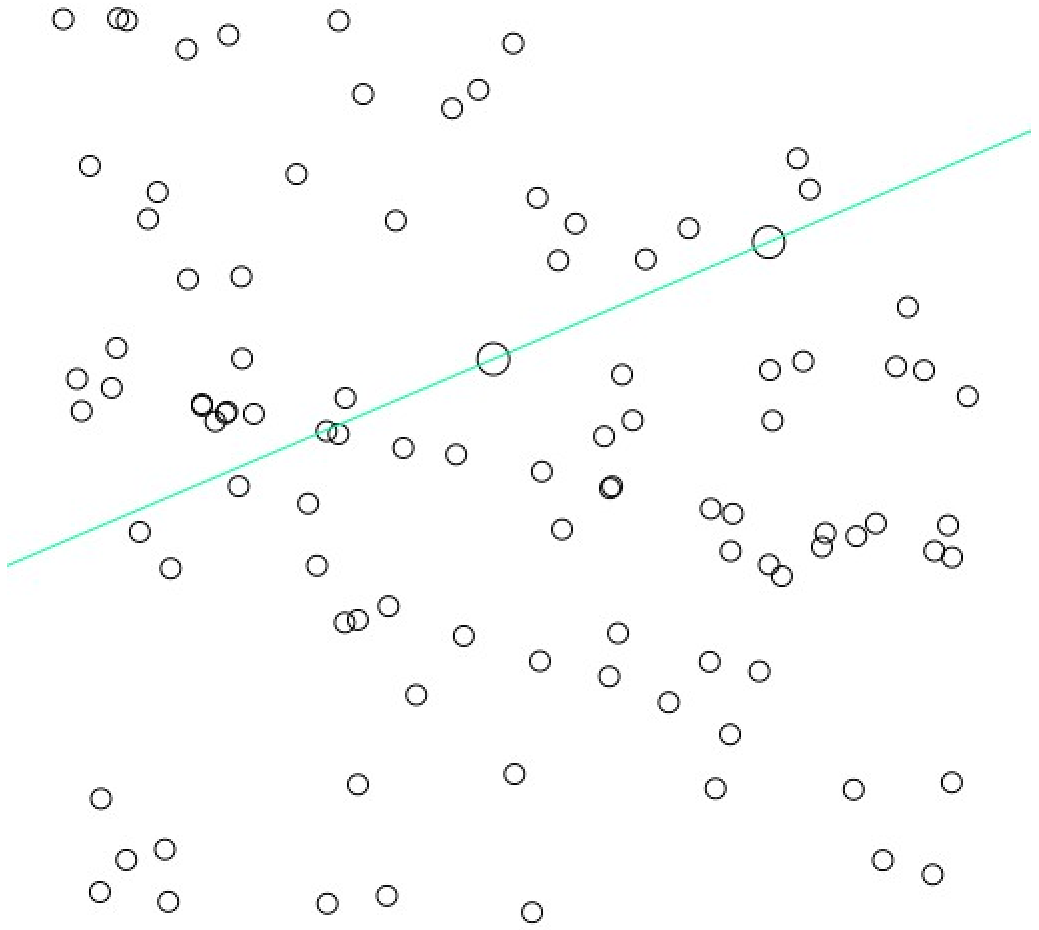
RANSAC



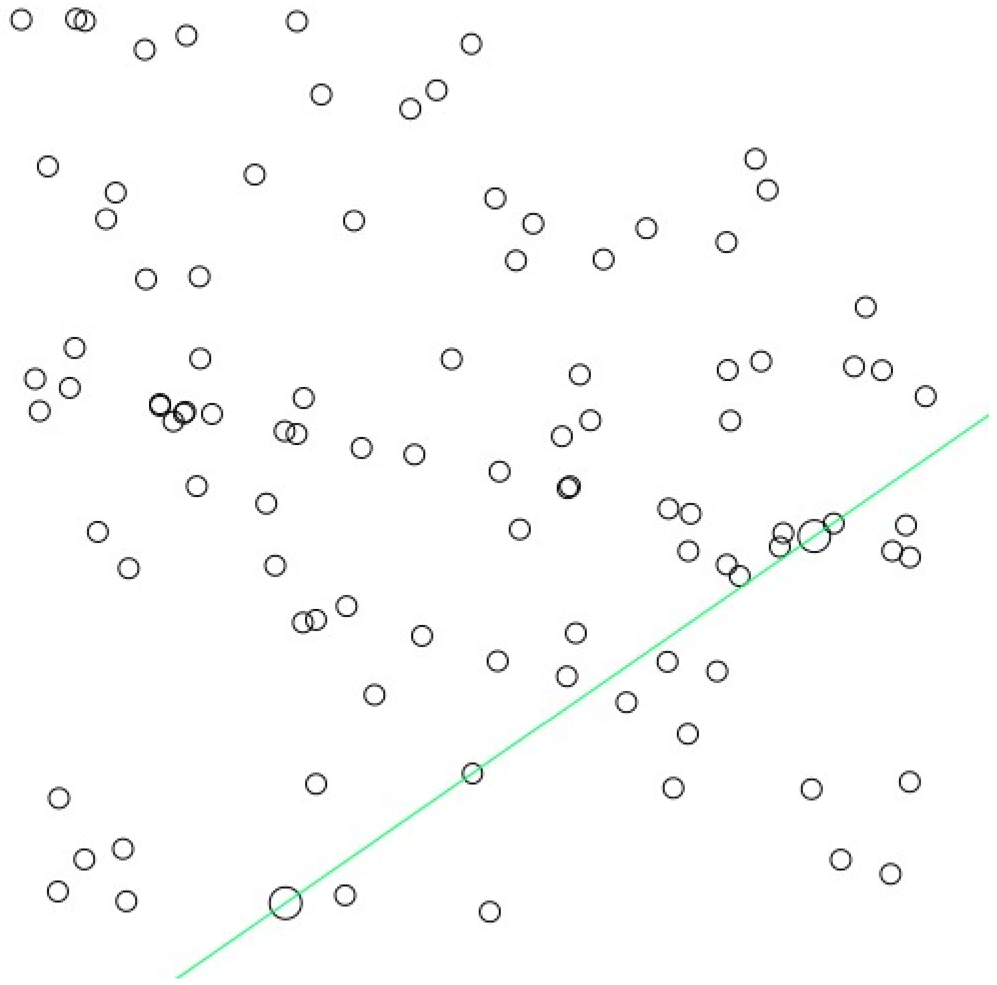
RANSAC



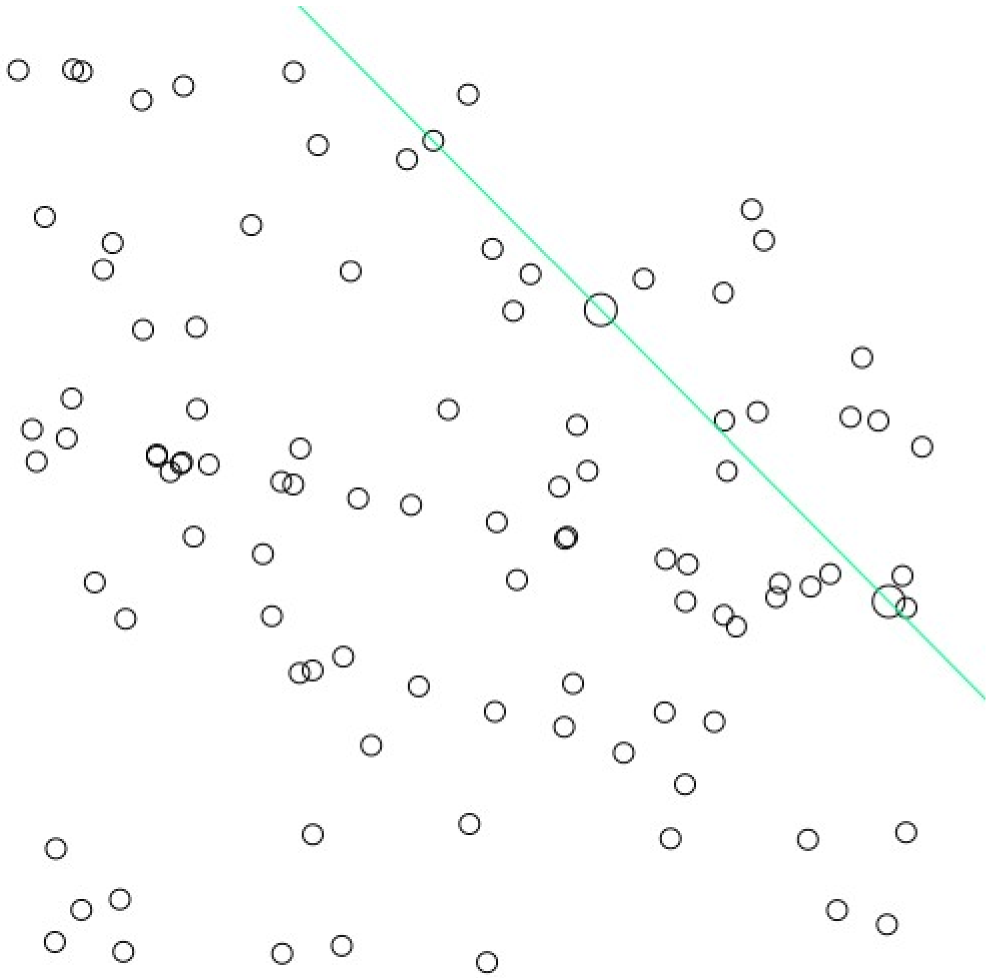
RANSAC



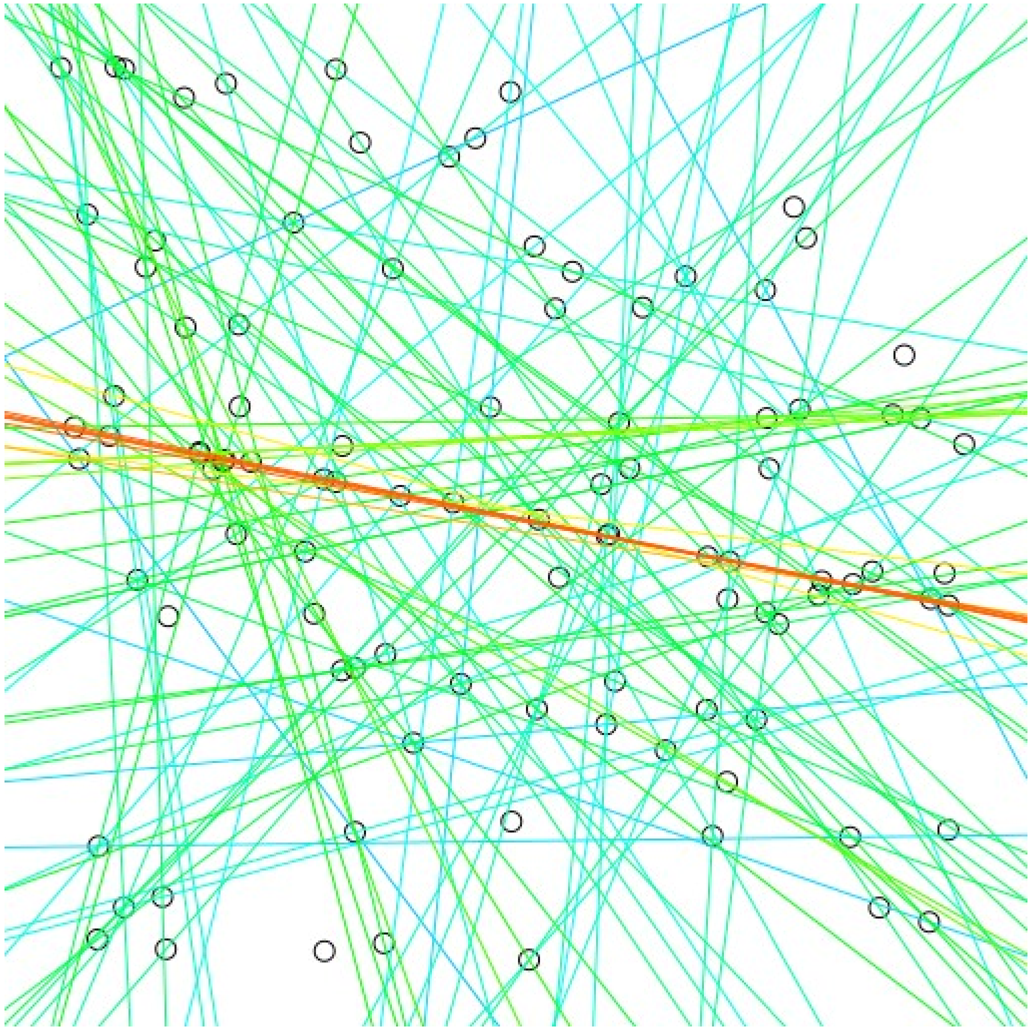
RANSAC



RANSAC



RANSAC



4. Exercise

6. Automatically Detect Point Pairs:

a) Detect keypoints and match descriptors
b) Filter matches based on ratio test and cross consistency check
c) Estimate Fundamental matrix
d) Compute average error, count number of inliers, and draw epipolar lines

7. Robust Estimation:

a) Implement RANSAC to estimate fundamental matrix robustly
b) Estimate fundamental matrix via RANSAC
c) Again compute average error, count number of inliers, and draw epipolar lines

4. Exercise - Theory

Two cameras c_1 and c_2 observe the same 3D scene in a general convergent constellation.

A human operator marked interesting points within a 3D scene. Two of those points are projected to $x_1 = (1,0,1)^T$ and $x_2 = (2,1,1)^T$ within the first view. What can you say about the position of those points within the second view, if the Fundamental matrix F of this system is given by

$$\boldsymbol{F} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -1 & 0 \\ 1 & -1 & 0 \end{bmatrix}$$

Using this information, derive the exact position of the epipole $\boldsymbol{e}_2$ within the second view.

How does the position of the epipole change, if the stereo-system is changed from a <u>convergent</u> to a <u>stereo-normal</u> view?

4. Exercise

Relative Orientation

Given:

- Main function for manual part (main_manual.cpp)
- Main function for automatic part (main_automatic.cpp)
- Unit tests (unit_test.cpp)
- Header of individual functions

Todo:

- Individual functions
    - Fill in the necessary function body parts (Pcv4.cpp)

Given

```
int getPointsManual(Mat &img1, Mat &img2,
                    vector<Vec3f>& p1, vector<Vec3f>& p2)
```

`img1`:	Data of first image
`img2`:	Data second image
`p1`:	Output: points in first image (homogeneous coordinates)
`p2`:	Output: points in second image (homogeneous coordinates)

- Displays the two images in two individual windows.
- Catches left mouse clicks, stores position in corresponding matrix and marks points in images by green circles.

```
void drawEpiLine(Mat& img, double a, double b, double c)
```

`img`:	Image in which lines are to be drawn
`a,b,c`:	Epipolar line in homogeneous representation

- Draws epipolar line into image

Given

`RawOrbMatches extractRawOrbMatches(Mat& img1, Mat& img2)`

`img1:` Data of first image

`img2:` Data second image

- Runs ORB (keypoint detector and feature extractor) on both images
- Matches the features (both ways)
- Returns keypoint locations and closest and second closest matches

```
struct RawOrbMatches {
    std::vector<cv::Vec3f> keypoints1; /// Location of keypoints in first image
    std::vector<cv::Vec3f> keypoints2; /// Location of keypoints in second image

    struct Match {
        unsigned closest;            /// Index of the closest match
        float closestDistance;       /// Feature distance of closest match
        unsigned secondClosest;      /// Index of the second closest match
        float secondClosestDistance; /// Feature distance of second closest match
    };

    /// For each keypoint in first images which keypoints are similar in second image
    std::map<unsigned, Match> matches_1_2;
    /// For each keypoint in second images which keypoints are similar in first image
    std::map<unsigned, Match> matches_2_1;
};
```

To Do

```
Matx33f getCondition2D(vector<Vec3f>& p)
```

$$T = \begin{bmatrix} \frac{1}{s_x} & 0 & \frac{-t_x}{s_x} \\ 0 & \frac{1}{s_y} & \frac{-t_y}{s_y} \\ 0 & 0 & 1 \end{bmatrix}$$

p:	Array of points
return:	The 3x3 matrix for point conditioning

- Computes the transformation matrix to move centroid to origin and scale mean distance to origin to one

```
vector<Vec3f> applyH_2D(vector<Vec3f>& geomObj,
                        Matx33f& H, GeometryType type)
```

geomObj	:	the geometric objects, that have to be transformed
H	:	the homography of the transformation
type	:	determines the type of the geometric object (point/line)
return	:	the transformed objects

- Reuse your implementation from exercise 1

To Do

```
Mat_<float> getDesignMatrix_fundamental(
                                    vector<Vec3f>& p1, vector<Vec3f>& p2)
```

p1:	Set of 2D points within first image
p2:	Set of 2D points within second image
return:	Output: Design matrix to compute fundamental matrix

- Takes corresponding point pairs (already conditioned)
- Generates design matrix to compute fundamental matrix

$$A=\begin{bmatrix} \hat{x}_1\hat{x}'_1 & \hat{y}_1\hat{x}'_1 & \hat{w}_1\hat{x}'_1 & \hat{x}_1\hat{y}'_1 & \hat{y}_1\hat{y}'_1 & \hat{w}_1\hat{y}'_1 & \hat{x}_1\hat{w}'_1 & \hat{y}_1\hat{w}'_1 & \hat{w}_1\hat{w}'_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \hat{x}_n\hat{x}'_n & \hat{y}_n\hat{x}'_n & \hat{w}_n\hat{x}'_n & \hat{x}_n\hat{y}'_n & \hat{y}_n\hat{y}'_n & \hat{w}_n\hat{y}'_n & \hat{x}_n\hat{w}'_n & \hat{y}_n\hat{w}'_n & \hat{w}_n\hat{w}'_n \end{bmatrix}$$

To Do

```
Mat_<float> getDesignMatrix_fundamental(
                    vector<Vec3f>& p1, vector<Vec3f>& p2)
```

p1:	Set of 2D points within first image
p2:	Set of 2D points within second image
return:	Output: Design matrix to compute fundamental matrix

- Takes corresponding point pairs (already conditioned)
- Generates design matrix to compute fundamental matrix

```
Matx33f solve_dlt_fundamental(Mat_<float>& A)
```

A:	Design matrix representing homogenous equation system $Af = 0$
return:	Fundamental matrix

- Uses SVD to solve homogeneous equation system

To Do

```
Matx33f forceSingularity(Matx33f& F)
```

F: Conditioned fundamental matrix (has to be transformed in-place)

Enforces rank 2 property of the fundamental matrix

$$\tilde{\boldsymbol{F}}^* = \tilde{\boldsymbol{U}}\ diag(d_1, d_2, 0)\, \tilde{\boldsymbol{V}}^T$$

To Do

```
Matx33f forceSingularity(Matx33f& F)
```

F: Conditioned fundamental matrix (has to be transformed in-place)

Enforces rank 2 property of the fundamental matrix

$$\tilde{F}^{*} = \tilde{U}\, diag(d_1, d_2, 0)\, \tilde{V}^{T}$$

```
Matx33f decondition_fundamental(
             Matx33f& T_fst, Matx33f& T_snd, Matx33f& F)
```

T_fst:	Conditioning matrix of set of 2D image points from first image
T_snd:	Conditioning matrix of set of 2D image points from second image
F:	Conditioned fundamental matrix

$$F = T'^{T} \tilde{F}^{*} T$$

- Deconditions fundamental matrix

To Do

```
Matx33f getFundamentalMatrix(vector<Vec3f>& p1,
                             vector<Vec3f>& p2)
```

p1: N points from first image
p1: N points from second image
return: Output: 3x3 fundamental matrix

- Estimates fundamental matrix from given set of point pairs
- Calls:

```
   getCondition2D(...)
   applyH_2d(...)
   getDesignMatrix_fundamental(...)
   solve_dlt_fundamental(...)
   forceSingularity(...)
   decondition_fundamental(...)
```

To Do

```
void visualize(cv::Mat& img1, cv::Mat& img2,
    vector<Vec3f>& p1, vector<Vec3f>& p2, cv::Matx33f& F)
```

img1:	Image data of first image
img2:	Image data of second image
p1:	Points in first image (homogeneous coordinates)
p2:	Points in second image (homogeneous coordinates)
F:	Calculated fundamental matrix

$$l' = F \cdot x$$

$$l = F^T \cdot x'$$

- Draws points
    - use cv::circle(...)
- Calculates and draws epipolar lines
    - use `drawEpiLine(...)`

To Do

```
float getError(Vec3f& p_fst, Vec3f& p_snd, Matx33f& F)
```

p_fst: Point in first image (homogeneous coordinates)
p_snd: Point in second image (homogeneous coordinates)
F: Calculated fundamental matrix

- Calculates Sampson errror for a single point pair

$$d = \frac{\left(\boldsymbol{x}_{\boldsymbol{i}}' \boldsymbol{F}\, \boldsymbol{x}_{\boldsymbol{i}}\right)^2}{\left(\boldsymbol{F}\, x_i\right)_a^2 + \left(\boldsymbol{F}\, x_i\right)_b^2 + \left(\boldsymbol{F}^{\boldsymbol{T}}\, \boldsymbol{x}_{\boldsymbol{i}}'\right)_a^2 + \left(\boldsymbol{F}^{\boldsymbol{T}}\, \boldsymbol{x}_{\boldsymbol{i}}'\right)_b^2}$$

To Do

```
float getError(vector<Vec3f>& p1, vector<Vec3f>& p2,
                                              Matx33f& F)
```

p1: Array of points in first image (homogeneous coordinates)
p2: Array of points in second image (homogeneous coordinates)
F: Calculated fundamental matrix

- Calculates the average Sampson error for all point pairs

$$d=\frac{1}{N}\sum_{i=1}^{N}\frac{\left(\mathbf{x}_i^{\prime}\mathbf{F}\,\mathbf{x}_i\right)^2}{\left(\mathbf{F}\,x_i\right)_a^2+\left(\mathbf{F}\,x_i\right)_b^2+\left(\mathbf{F}^T\,\mathbf{x}_i^{\prime}\right)_a^2+\left(\mathbf{F}^T\,\mathbf{x}_i^{\prime}\right)_b^2}$$

To Do

```
float getError(vector<Vec3f>& p1, vector<Vec3f>& p2,
                                                                                        Matx33f& F)
```

p1:	Array of points in first image (homogeneous coordinates)
p2:	Array of points in second image (homogeneous coordinates)
F:	Calculated fundamental matrix

- Calculates the average Sampson error for all point pairs

```
unsigned countInliers(vector<Vec3f>& p1,
                      vector<Vec3f>& p2,
                      Matx33f& F, float threshold)
```

p1:	Points in first image (homogeneous coordinates)
p2:	Points in second image (homogeneous coordinates)
F:	Calculated fundamental matrix
threshold:	Maximal Sampson error of an inlier

- Count pairs with Sampson error smaller than threshold

To Do

```
Matx33f estimateFundamentalRANSAC(vector<Vec3f>& p
     vector<Vec3f>& p2, unsigned numIters, float thresh)
```

`p1`: Points in first image (homogeneous coordinates)

`p2`: Points in second image (homogeneous coordinates)

`numIters`: Number of attempts

`thresh`: Threshold on Sampson error for inliers

```
- for numIters:
    - pick a random subset of 8 points
    - estimate fundamental matrix of subset
    - count total number of inliers

- return fundamental matrix with most inliers
```

Code snipped for random number generation in source code.

To Do

```
void filterMatches(RawOrbMatches &rawOrbMatches,
                   vector<Vec3f>& p1, vector<Vec3f>& p2)
```

rawOrbMatches: keypoint locations and closest and second closest matches

p1: Output: points in first image (homogeneous coordinates)

p2: Output: points in second image (homogeneous coordinates)

- Loops over raw matches and extracts those that satisfy:
  - Ratio test (ratio of 0.75)
  - Cross Consistency Check

- Pairs returned in p1 and p2

To Do

```
void getPointsAutomatic(Mat &img1, Mat &img2,
                        vector<Vec3f>& p1, vector<Vec3f>& p2)
```

img1:	Data of first image
img2:	Data second image
p1:	Output: points in first image (homogeneous coordinates)
p2:	Output: points in second image (homogeneous coordinates)

- Runs extraction, matching, and filtering
- Calls:
    - extractRawOrbMatches(…)
    - filterMatches(...)

- Optional: Visualize with drawMatches(…) (in Helper.h/.cpp)

Midterm

Available **From**: Tuesday 12.01.2021 ~12:00
Until: Tuesday 19.01.2021 23.59

- 3 attempts
  - ➔ Can be taken anytime during the available period
- Best attempt is taken as final result
- Multiple Choice
  - ➔ Multiple correct answers possible
- 45 min
- 50% of points are required for pass
- Wrong choices give -0.5 Points
  - ➔ You can get negative points on a question
  - ➔ Only select answers if you are confident