Institut für Kartographie und Geoinformatik | Leibniz Universität Hannover

Estimation of transformations Here: Similarity transform

Claus.Brenner@ikg.uni-hannover.de

Leibniz Universität Hannover

Estimation of transformations

- Here: similarity transformation
- Reason:
  - A frequent task is to align two (or more) independently acquired datasets into one common coordinate system
  - In that case, every single dataset is not “deformed” but rather
    - rotated
    - shifted
    - (possibly) scaled
- A similarity transform can also be used to align a model (of an object) with a scene (e.g. a scan)
  - Object recognition and object reconstruction
  - E.g. Model: Cylinder; Scene: scan of an industrial facility.

Similarity transform

- Formulation in terms of matrices:

$$\mathbf{r} = s\mathbf{R}\mathbf{l} + \mathbf{t}$$

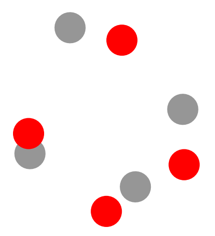
- …where:
  - $s \in \mathbb{R}$ scale
  - $\mathbf{R}$ 3x3 rotation matrix
  - $\mathbf{t} \in \mathbb{R}^3$ translation vector
  - $\mathbf{l}$ point in “left” scene
  - $\mathbf{r}$ transformed point, point in “right” scene.

Estimation of a similarity transform

- Sought for: Estimation of the parameters of a similarity transform
- Given: point correspondences between “right” and “left” points
- I.e., we are looking for the parameters of a transformation such that:

$$\mathbf{r}_i = s\mathbf{R}\mathbf{l}_i + \mathbf{t}$$

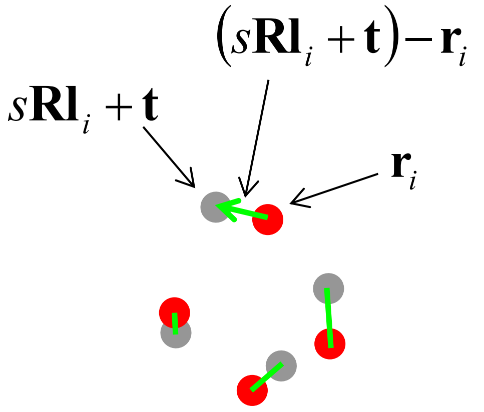
for all i = 1,…,n (number of point correspondences)



Estimation of a similarity transform

- In reality, measurements are subject to measurement errors
- Instead of a perfect fit, we are searching for a solution which minimizes a cost function
- E.g. least squares minimization: search for the parameters of the transformation which fulfil:

$$\sum_i \left\| \left(s\mathbf{R}\mathbf{l}_i + \mathbf{t} \right) - \mathbf{r}_i \right\|^2 \overset{!}{=} \min$$



$\left(s\mathbf{R}\mathbf{l}_i + \mathbf{t} \right) - \mathbf{r}_i$
$s\mathbf{R}\mathbf{l}_i + \mathbf{t}$
$\mathbf{r}_i$

Estimation of a similarity transform

- We are looking for 7 parameters:
  - Scale (1)
  - Translation (3)
  - Rotation (3)

- First, let us look at the rotation only
  - Reason: see later
  - I.e., assume for now:

$$\mathbf{t} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^{\mathrm{T}}$$

$$s = 1$$

Estimation of the rotational part

- The function to be minimized is then:

$$\sum_i \left\| \mathbf{R}\mathbf{l}_i - \mathbf{r}_i \right\|^2 \overset{!}{=} \min$$

- Instead of minimizing the squared absolute value, one can equivalently maximize the scalar product:

$$\begin{aligned}
\underbrace{\sum_i \left\| \mathbf{R}\mathbf{l}_i - \mathbf{r}_i \right\|^2}_{\min} &= \sum_i \left\langle \mathbf{R}\mathbf{l}_i - \mathbf{r}_i, \mathbf{R}\mathbf{l}_i - \mathbf{r}_i \right\rangle \\
&= \sum_i \left\langle \mathbf{R}\mathbf{l}_i, \mathbf{R}\mathbf{l}_i \right\rangle - 2\sum_i \left\langle \mathbf{R}\mathbf{l}_i, \mathbf{r}_i \right\rangle + \sum_i \left\langle \mathbf{r}_i, \mathbf{r}_i \right\rangle \\
&= \text{const.} - 2\underbrace{\sum_i \left\langle \mathbf{R}\mathbf{l}_i, \mathbf{r}_i \right\rangle}_{\max} + \text{const.}
\end{aligned}$$

Estimation of the rotational part

- If $\mathbf{R}$ is to be determined in the "usual" way, one substitutes the elementary rotation matrices for it and then searches for a maximum of….

$$\sum_i \langle \mathbf{R}\mathbf{l}_i, \mathbf{r}_i \rangle = \sum_i \langle \mathbf{R}(\omega,\varphi,\kappa)\mathbf{l}_i, \mathbf{r}_i \rangle = f(\omega,\varphi,\kappa)$$

- …by determining the partial derivatives and searching for a root
- The function and its partial derivatives contain transcendent (especially: non-linear) functions
- The solution is found using iterations
  - Initial values are necessary
  - If the initial values are bad, the global maximum may be missed.

Rotation estimation – quaternions

- Using quaternions, the equation is:

$$f(q) = \sum_i \langle \mathbf{R}\mathbf{l}_i, \mathbf{r}_i \rangle = \sum_i \langle q l_i q^*, r_i \rangle$$

- Where
  - we search for the unit quaternion *q*, which represents the sought-for rotation (maximizing *f(q)*)
  - l_i and r_i are the "quaternion representations" of the points, i.e.

$$l_i = \begin{bmatrix} 0 \\ l_{i,1} \\ l_{i,2} \\ l_{i,3} \end{bmatrix} = \begin{bmatrix} 0 \\ x_{i,\text{left}} \\ y_{i,\text{left}} \\ z_{i,\text{left}} \end{bmatrix} \qquad r_i = \begin{bmatrix} 0 \\ r_{i,1} \\ r_{i,2} \\ r_{i,3} \end{bmatrix} = \begin{bmatrix} 0 \\ x_{i,\text{right}} \\ y_{i,\text{right}} \\ z_{i,\text{right}} \end{bmatrix}$$

Expansion of *f(q)*

- Use the following trick: for the scalar product of quaternions, it holds that:

$$\left\langle q l q^{*}, r \right\rangle = \left\langle q l, r q \right\rangle$$

  - From this, it follows:

$$f(q) = \sum_{i} \left\langle q l_i q^{*}, r_i \right\rangle = \sum_{i} \left\langle q l_i, r_i q \right\rangle$$

Proof for the “trick”

- Note:

$$\left(\hat{\mathbf{T}}_{q^*}\right)^{\mathrm{T}} = \begin{bmatrix} q_0 & \mathbf{q}^{\mathrm{T}} \\ -\mathbf{q} & q_0\mathbf{I}+\mathbf{S}_{\mathbf{q}} \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} q_0 & -\mathbf{q}^{\mathrm{T}} \\ \mathbf{q} & q_0\mathbf{I}-\mathbf{S}_{\mathbf{q}} \end{bmatrix} = \hat{\mathbf{T}}_q$$

- From this it follows:

$$\begin{aligned} \left\langle a\cdot q^*, b\right\rangle &= \left\langle \hat{\mathbf{T}}_{q^*}a, b\right\rangle && \text{(quat. multiplication} \rightarrow \text{matrix mult.)} \\ &= \left(\hat{\mathbf{T}}_{q^*}a\right)^{\mathrm{T}} b = a^{\mathrm{T}}\left(\hat{\mathbf{T}}_{q^*}\right)^{\mathrm{T}} b && \text{(def. scalar product / transposition)} \\ &= a^{\mathrm{T}}\hat{\mathbf{T}}_q b && \text{(equation from above)} \\ &= a^{\mathrm{T}}\left(b\cdot q\right) && \text{(matrix mult.} \rightarrow \text{quat. mult.)} \\ &= \left\langle a, b\cdot q\right\rangle && \text{(def. scalar product)} \end{aligned}$$

Expansion of *f(q)*

- Switch to representation of quaternion multiplication in terms of matrices
  - Where: not *q* is expressed by a matrix, but rather *l* and *r* !!

$$q l_i = \hat{\mathbf{T}}_{l_i}\mathbf{q} \qquad r_i q = \mathbf{T}_{r_i}\mathbf{q}$$

- From this, it follows:

$$\begin{aligned} f(q) &= \sum_i \langle q l_i, r_i q \rangle \\ &= \sum_i \left\langle \hat{\mathbf{T}}_{l_i}\mathbf{q}, \mathbf{T}_{r_i}\mathbf{q} \right\rangle \\ &= \sum_i \mathbf{q}^{\mathrm{T}} \hat{\mathbf{T}}_{l_i}^{\mathrm{T}} \mathbf{T}_{r_i}\mathbf{q} \\ &= \mathbf{q}^{\mathrm{T}} \left(\sum_i \hat{\mathbf{T}}_{l_i}^{\mathrm{T}} \mathbf{T}_{r_i} \right) \mathbf{q} \end{aligned}$$

...the solution

- We are looking for the unit quaternion which maximizes the quadratic form:

$$\mathbf{q}^{\mathrm{T}}\left(\sum_i \hat{\mathbf{T}}_{l_i}^{\mathrm{T}}\mathbf{T}_{r_i}\right)\mathbf{q}$$

- Note that for any symmetric matrix $\mathbf{S}$:

$$\arg\max_{\|\mathbf{x}\|=1} \mathbf{x}^{\mathrm{T}}\mathbf{S}\mathbf{x} = \mathbf{e}_1$$

  where $\mathbf{e}_1$ is the eigenvector of the largest eigenvalue of $\mathbf{S}$.

- I.e., the sought-after rotation is the quaternion, which is given by the eigenvector belonging to the largest eigenvalue of:

$$\sum_i \hat{\mathbf{T}}_{l_i}^{\mathrm{T}}\mathbf{T}_{r_i}$$

Practical implementation

- Given: point pairs (l_i, r_i)
  - For every pair l, r, set up the two matrices

$$\hat{\mathbf{T}}_l = \begin{bmatrix} 0 & -x_l & -y_l & -z_l \\ x_l & 0 & z_l & -y_l \\ y_l & -z_l & 0 & x_l \\ z_l & y_l & -x_l & 0 \end{bmatrix} \qquad \mathbf{T}_r = \begin{bmatrix} 0 & -x_r & -y_r & -z_r \\ x_r & 0 & -z_r & y_r \\ y_r & z_r & 0 & -x_r \\ z_r & -y_r & x_r & 0 \end{bmatrix}$$

  - Compute the 4x4 matrix

$$\hat{\mathbf{T}}_l^{\mathrm{T}} \mathbf{T}_r$$

  - Sum up over all matrices (one element for every point pair)

$$\mathbf{S} = \sum_i \hat{\mathbf{T}}_{l_i}^{\mathrm{T}} \mathbf{T}_{r_i}$$

  - The required rotation is the eigenvector belonging to the largest eigenvalue

How to compute the matrix **S**

$$\hat{\mathbf{T}}_l^{\mathrm{T}} = \begin{bmatrix} 0 & x_l & y_l & z_l \\ -x_l & 0 & -z_l & y_l \\ -y_l & z_l & 0 & -x_l \\ -z_l & -y_l & x_l & 0 \end{bmatrix} \qquad \mathbf{T}_r = \begin{bmatrix} 0 & -x_r & -y_r & -z_r \\ x_r & 0 & -z_r & y_r \\ y_r & z_r & 0 & -x_r \\ z_r & -y_r & x_r & 0 \end{bmatrix}$$

$$\hat{\mathbf{T}}_l^{\mathrm{T}} \mathbf{T}_r = \begin{bmatrix} x_l x_r + y_l y_r + z_l z_r & y_l z_r - z_l y_r & \ldots & \ldots \\ y_l z_r - z_l y_r & x_l x_r - y_l y_r - z_l z_r & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots \end{bmatrix}$$

(symmetric)

How to compute the matrix **S**

- By defining the following 9 sums:

$$S_{xx} = \sum_{i=1}^{n} x_{l,i} x_{r,i} \qquad S_{xy} = \sum_{i=1}^{n} x_{l,i} y_{r,i} \qquad S_{xz} = \sum_{i=1}^{n} x_{l,i} z_{r,i} \qquad \ldots$$

one obtains:

$$\mathbf{S} = \begin{bmatrix} (S_{xx}+S_{yy}+S_{zz}) & S_{yz}-S_{zy} & S_{zx}-S_{xz} & S_{xy}-S_{yx} \\ . & (S_{xx}-S_{yy}-S_{zz}) & S_{xy}+S_{yx} & S_{zx}+S_{xz} \\ . & . & (-S_{xx}+S_{yy}-S_{zz}) & S_{yz}+S_{zy} \\ . & . & . & (-S_{xx}-S_{yy}+S_{zz}) \end{bmatrix}$$

How to compute the matrix **S**

- Therefore, we obtain the following simplified computation:
  - For every point pair *l_i*, *r_i* :
    - Using the 3 coordinates of each vector, form all (9) possible products:

      $$x_{l,i}x_{r,i} \quad x_{l,i}y_{r,i} \quad x_{l,i}z_{r,i} \qquad y_{l,i}x_{r,i} \quad y_{l,i}y_{r,i} \quad \ldots$$

    - Add them up in the 9 sum terms:

      $$S_{xx} \quad S_{xy} \quad S_{xz} \qquad S_{yx} \quad S_{yy} \quad \ldots$$

  - After iterating over all point pairs, set up the (symmetric) Matrix **S**, as shown on the previous slide
  - Get the eigenvector belonging to the largest eigenvalue of **S**, which is the quaternion which represents the required rotation
  - If required, convert the quaternion into a 3x3 rotation matrix.

Translation and Scale

Translation and scale

- So far, we assumed:
  - No translation: $\mathbf{t} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^{\mathrm{T}}$
  - No scaling: $s = 1$
- Under these conditions, we have found the optimal (least squares) solution for the rotation
- Now: solve the original problem (7 parameters):

$$\sum_i \left\| \left(s\mathbf{R}\mathbf{l}_i + \mathbf{t} \right) - \mathbf{r}_i \right\|^2 \overset{!}{=} \min$$

Translation

- First compute center of mass of each point set:

$$\bar{\mathbf{l}} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{l}_i \qquad \bar{\mathbf{r}} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{r}_i$$

- Use this to compute reduced (by the center of mass) coordinates:

$$\mathbf{l}'_i = \mathbf{l}_i - \bar{\mathbf{l}} \qquad \mathbf{r}'_i = \mathbf{r}_i - \bar{\mathbf{r}}$$

- Trivially, for the reduced coordinates the following holds:

$$\sum_{i=1}^{n} \mathbf{l}'_i = \mathbf{0} \qquad \sum_{i=1}^{n} \mathbf{r}'_i = \mathbf{0}$$

Translation

- It follows for the error term:

$$\begin{aligned} s\mathbf{R}\mathbf{l}_i + \mathbf{t} - \mathbf{r}_i &= s\mathbf{R}(\mathbf{l}'_i + \bar{\mathbf{l}}) + \mathbf{t} - (\mathbf{r}'_i + \bar{\mathbf{r}}) \\ &= s\mathbf{R}\mathbf{l}'_i - \mathbf{r}'_i + \underbrace{\mathbf{t} + s\mathbf{R}\bar{\mathbf{l}} - \bar{\mathbf{r}}}_{\mathbf{t}'} \\ &= s\mathbf{R}\mathbf{l}'_i - \mathbf{r}'_i + \mathbf{t}' \end{aligned}$$

- Where we defined:

$$\mathbf{t}' = \mathbf{t} + s\mathbf{R}\bar{\mathbf{l}} - \bar{\mathbf{r}}$$

Which is the translation when using reduced coordinates

Translation

- Starting from the least squares sum:

$$\sum_i \left\| s\mathbf{R}\mathbf{l}'_i - \mathbf{r}'_i + \mathbf{t}' \right\|^2$$

- We multiply this out (binomial identity):

$$= \sum_i \left\| s\mathbf{R}\mathbf{l}'_i - \mathbf{r}'_i \right\|^2 + 2\mathbf{t}'^{\mathrm{T}} \sum_i \left(s\mathbf{R}\mathbf{l}'_i - \mathbf{r}'_i \right) + n\left\| \mathbf{t}' \right\|^2$$

- Observation:
  - The middle term is 0 (since $\sum_{i=1}^{n} \mathbf{l}'_i = \mathbf{0}$ and $\sum_{i=1}^{n} \mathbf{r}'_i = \mathbf{0}$)
- Therefore, what remains is:

$$= \sum_i \left\| s\mathbf{R}\mathbf{l}'_i - \mathbf{r}'_i \right\|^2 + n\left\| \mathbf{t}' \right\|^2$$

Translation

$$= \underbrace{\sum_i \left\| s\mathbf{R}\mathbf{l}'_i - \mathbf{r}'_i \right\|^2}_{1} + \underbrace{n\left\|\mathbf{t}'\right\|^2}_{2}$$

- For the two remaining terms, it can be observed:
  - The first term is independent of $\mathbf{t}'$
  - The second term cannot be negative
- Therefore, the minimum is obtained for

  $$\left\|\mathbf{t}'\right\|^2 = 0$$
- I.e., the least squares solution is:

  $$\mathbf{t}' = \mathbf{0}$$
- Which, by the definition of $\mathbf{t}'$ leads to:

  $$\mathbf{t} + s\mathbf{R}\bar{\mathbf{l}} - \bar{\mathbf{r}} = \mathbf{0}$$

Translation: Solution

- Independent of the optimization of **R** and *s*, the optimal translation is:

$$\boxed{\mathbf{t} = \bar{\mathbf{r}} - s\mathbf{R}\bar{\mathbf{l}}}$$

- **I.e., the translation is the difference between the center of mass of the right points and the scaled and rotated center of mass of the left points**

- The remaining least squares error to minimize (now, independent of **t**) is:

$$\sum_i \left\| s\mathbf{R}\mathbf{l}'_i - \mathbf{r}'_i \right\|^2$$

Scale

- Multiply out the remaining term:

$$\sum_i \left\| s\mathbf{R}\mathbf{l}'_i - \mathbf{r}'_i \right\|^2$$

$$= \sum_i \left\| s\mathbf{R}\mathbf{l}'_i \right\|^2 - 2s\sum_i (\mathbf{r}'^{\mathrm{T}}_i \mathbf{R}\mathbf{l}'_i) + \sum_i \left\| \mathbf{r}'_i \right\|^2$$

$$= s^2 \sum_i \left\| \mathbf{l}'_i \right\|^2 - 2s\sum_i (\mathbf{r}'^{\mathrm{T}}_i \mathbf{R}\mathbf{l}'_i) + \sum_i \left\| \mathbf{r}'_i \right\|^2$$

$$= s^2 S_l - 2sD + S_r$$

- Completing the square:

$$= \left(\sqrt{S_l}\, s - D / \sqrt{S_l} \right)^2 - D^2 / S_l + S_r$$

Scale

- Minimum is attained for "quadratic term = 0":

$$\underbrace{\left(\sqrt{S_l}\,s - D/\sqrt{S_l}\right)^2}_{=0} - D^2/S_l + S_r$$

- I.e.:

$$\boxed{s = \frac{D}{S_l} = \frac{\sum_i \mathbf{r}_i'^{\mathrm{T}} \mathbf{R} \mathbf{l}_i'}{\sum_i \left\|\mathbf{l}_i'\right\|^2}}$$

  - These are the scalar products of the right with the rotated left vectors, divided by the squared lengths of the left vectors

Scale – Symmetry

- Note: when using this approach, the formula for the scale is not symmetric:

$$s = \frac{\sum_i \mathbf{r}_i'^{\mathrm{T}} \mathbf{R} \mathbf{l}_i'}{\sum_i \left\| \mathbf{l}_i' \right\|^2}$$

- If instead we formulate the scale in a symmetric way:

$$\sum_i \left\| \sqrt{s} \mathbf{R} \mathbf{l}_i' - \frac{1}{\sqrt{s}} \mathbf{r}_i' \right\|^2$$

- Then the following symmetric formula is obtained (which in addition does not need **R**):

$$\boxed{s = \left(\frac{\sum_i \left\| \mathbf{r}_i' \right\|^2}{\sum_i \left\| \mathbf{l}_i' \right\|^2} \right)^{1/2}}$$

Remaining: Rotation

- Note: *s* will be always selected such that the first term will be zero (for any given rotation):

$$\underbrace{\left(\sqrt{S_l}\,s - D/\sqrt{S_l}\right)^2}_{=0} - D^2/S_l + S_r$$

- The entire term is therefore minimal, if D is maximal

$$D = \sum_i (\mathbf{r}_i'^{\mathrm{T}} \mathbf{R} \mathbf{l}_i') \overset{!}{=} \max$$

- This is exactly the scalar product which we have maximized in the beginning (this time, it uses reduced coordinates):

$$D = \sum_i \langle \mathbf{R}\mathbf{l}_i', \mathbf{r}_i' \rangle$$

Cookbook: Estimation of a similarity transform from point correspondences

Given: two point lists $\mathbf{l}_i\,,\mathbf{r}_i\,,\; i \in 1,2,\ldots,n$

Sought for: scale $s \in \mathbb{R}$, rotation $\mathbf{R}$, Translation $\mathbf{t} \in \mathbb{R}^3$ which minimize the following objective function:

$$\sum_i \left\| \left(s\mathbf{R}\mathbf{l}_i + \mathbf{t} \right) - \mathbf{r}_i \right\|^2 \overset{!}{=} \min$$

Steps:

1. Reduce both point lists by their center of mass
2. Compute the rotation
3. Compute the scale
4. Compute the translation
5. If required, apply the transformation to one of the point lists (usually, the left list).

Cookbook: Estimation of a similarity transform from point correspondences – details

1. Reduction by the center of mass

$$\bar{\mathbf{l}} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{l}_i \qquad \bar{\mathbf{r}} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{r}_i \quad \Rightarrow \quad \mathbf{l}'_i = \mathbf{l}_i - \bar{\mathbf{l}} \qquad \mathbf{r}'_i = \mathbf{r}_i - \bar{\mathbf{r}}$$

2. Computation of the rotation

$$S_{xx} = \sum_{i=1}^{n} x'_{l,i} x'_{r,i} \qquad S_{xy} = \sum_{i=1}^{n} x'_{l,i} y'_{r,i} \qquad S_{xz} = \sum_{i=1}^{n} x'_{l,i} z'_{r,i} \qquad \ldots$$

$$\Rightarrow \quad \mathbf{S} = \begin{bmatrix} (S_{xx}+S_{yy}+S_{zz}) & S_{yz}-S_{zy} & S_{zx}-S_{xz} & S_{xy}-S_{yx} \\ . & (S_{xx}-S_{yy}-S_{zz}) & S_{xy}+S_{yx} & S_{zx}+S_{xz} \\ . & . & (-S_{xx}+S_{yy}-S_{zz}) & S_{yz}+S_{zy} \\ . & . & . & (-S_{xx}-S_{yy}+S_{zz}) \end{bmatrix}$$

$\Rightarrow$ *q* eigenvector of the largest eigenvalue of **S**: Quaternion

$\Rightarrow$ **R** rotation matrix, computed from *q* (see last lecture)

Cookbook: Estimation of a similarity transform from point correspondences – details

3. Computation of the (symmetric) scale factor

$$s = \left(\frac{\sum_i \left\| \mathbf{r}'_i \right\|^2}{\sum_i \left\| \mathbf{l}'_i \right\|^2} \right)^{1/2}$$

4. Computation of the translation
   - This requires the (already computed) scale and rotation

$$\mathbf{t} = \bar{\mathbf{r}} - s\mathbf{R}\bar{\mathbf{l}}$$

5. Application of the transformation, if required

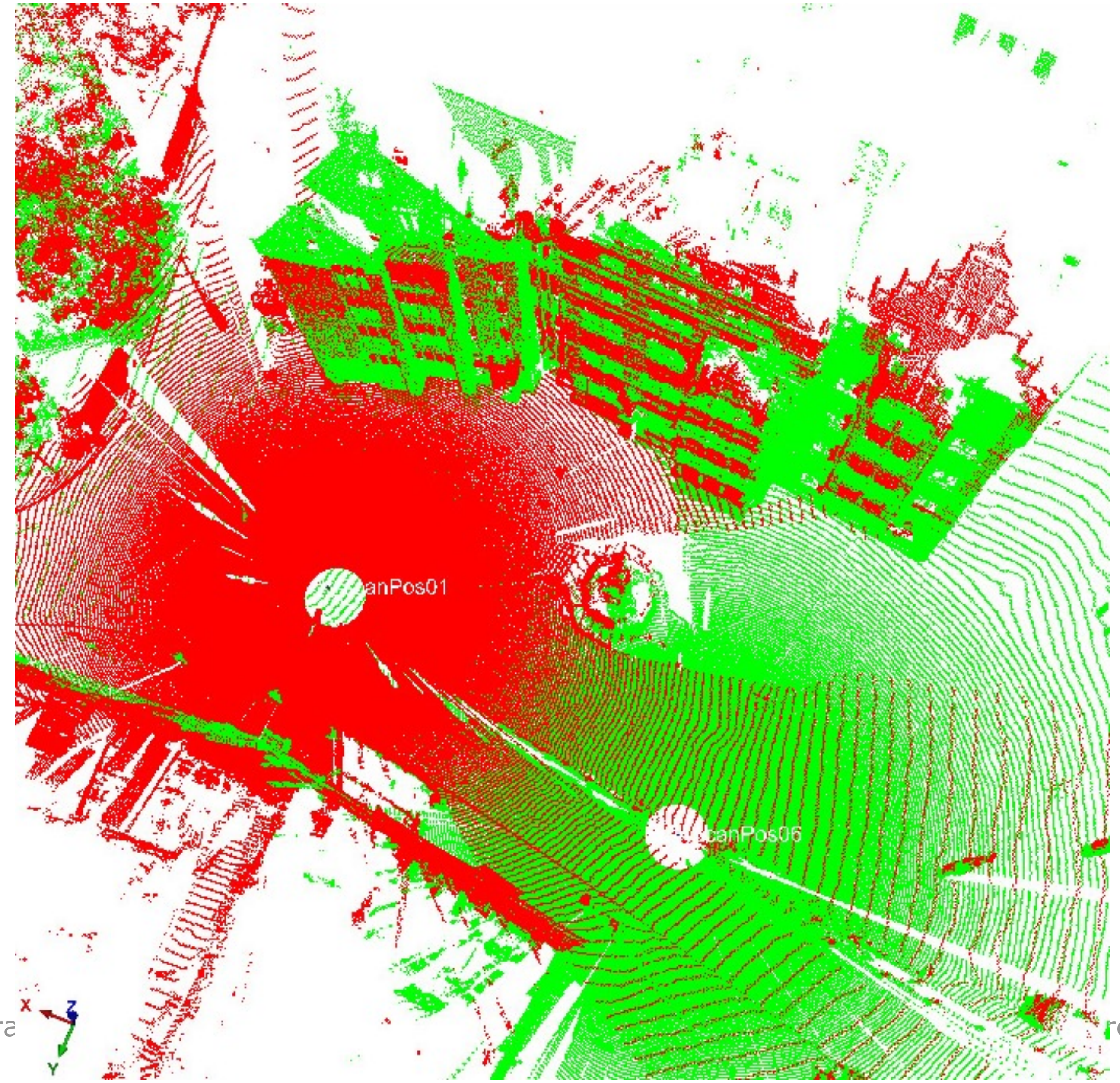
$$\tilde{\mathbf{l}}_i = s\mathbf{R}\mathbf{l}_i + \mathbf{t}$$

Estimating a transform when correspondences are unknown

Estimation with unknown correspondences

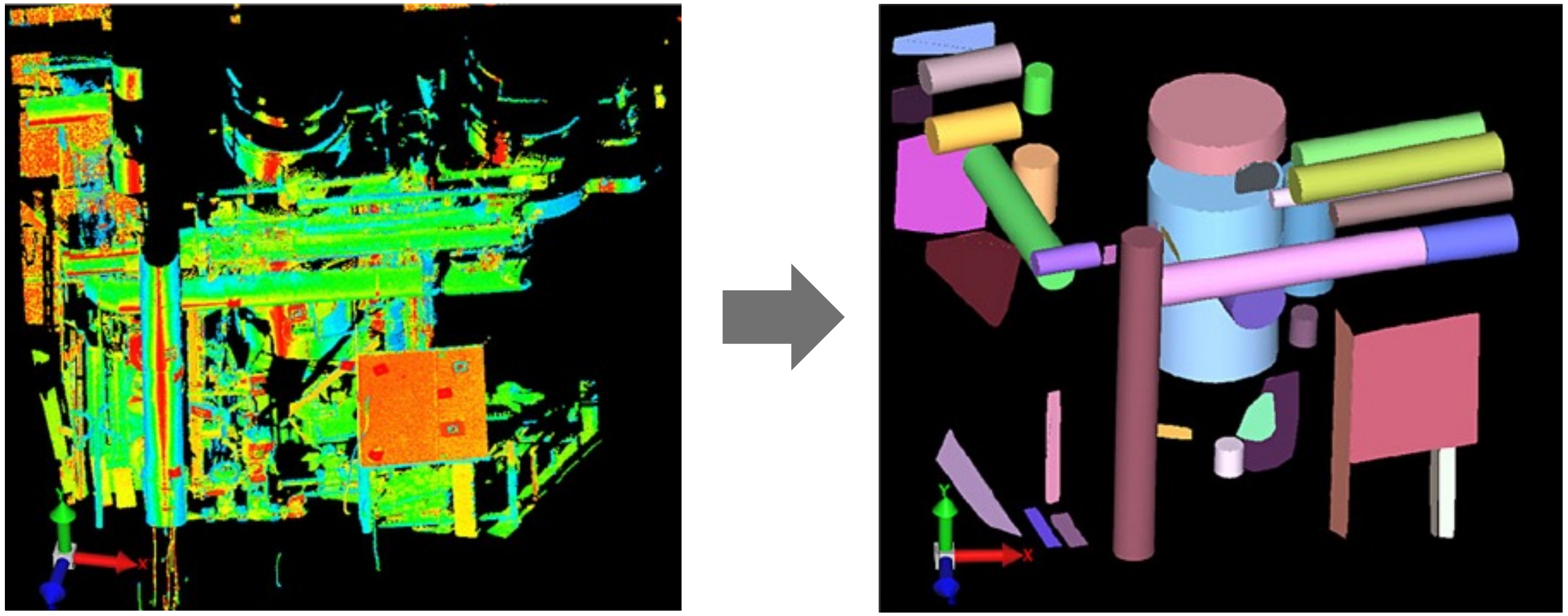
- So far:
  - Left and right point list
  - Assumption: point $\mathbf{l}_i$, $\mathbf{r}_i$ describe the same object point
- This is a typical situation for “classical surveying applications”
- Signalized points, point numbers are introduced, a sketch is drawn, points are measured…
- Nowadays:
  - A terrestrial scanner measures 400M points
  - It is taken to another scan position
  - There, it again measures 400M points
  - The point clouds shall be aligned to each other (where they overlap)

Estimation with unknown correspondences



Estimation with unknown correspondences

- Another application: scene – model assignment
  - An industrial facility is scanned
  - Standardized industrial parts are held in a database (pipes, beams, connecting elements…)
  - The position and orientation of the objects in the scene shall be determined

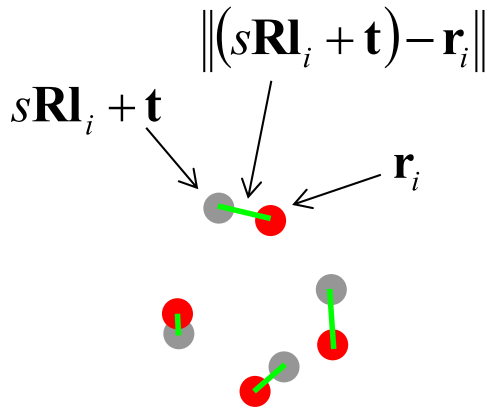


[Rabbani, Dijkman, van den Heuvel, Vosselman, ISPRSJ 61 (2007) 355–370]

Optimization problem

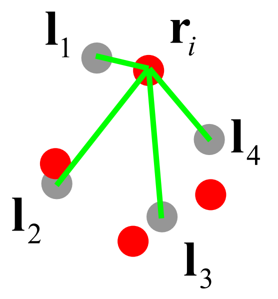
- So far: known correspondences: $\mathbf{r}_i \leftrightarrow \mathbf{l}_i$

$$(s^*, \mathbf{R}^*, \mathbf{t}^*) = \arg\min_{s,\mathbf{R},\mathbf{t}} \sum_i \left\| (s\mathbf{R}\mathbf{l}_i + \mathbf{t}) - \mathbf{r}_i \right\|^2$$


$\left\| (s\mathbf{R}\mathbf{l}_i + \mathbf{t}) - \mathbf{r}_i \right\|$
$s\mathbf{R}\mathbf{l}_i + \mathbf{t}$
$\mathbf{r}_i$

- Now: correspondences to be established: $\mathbf{r}_i \leftrightarrow \mathbf{l}_j, \quad j = n(i)$

$$(s^*, \mathbf{R}^*, \mathbf{t}^*, n^*) = \arg\min_{s,\mathbf{R},\mathbf{t},n} \sum_i \left\| (s\mathbf{R}\mathbf{l}_{n(i)} + \mathbf{t}) - \mathbf{r}_i \right\|^2$$



Where n^* maps each index i to an index j.

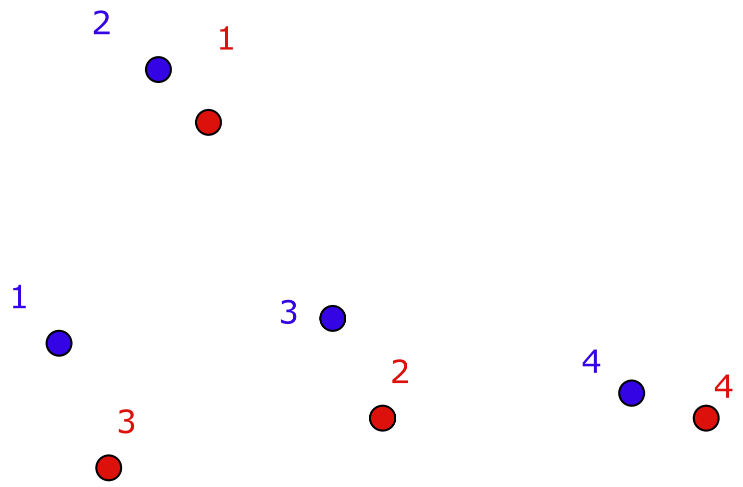
Problem: How many such mappings n are there?

The „Iterative Closest Point‟ (ICP) Algorithm

- Basic idea:
  - Assume sufficiently good initial values for the transformation
  - Assume that “close” points correspond to each other, establish point assignments
  - Estimate a new transformation using these assignments
  - Repeat: using the new transformation, establish new assignments, and estimate transformation again
  - Iterate this until the transformation does not change anymore

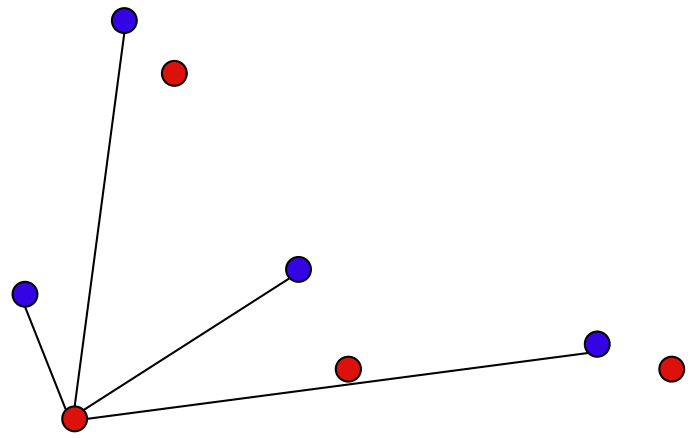
Iterative Closest Point Algorithm

- Two point sets (red, blue)
- Correspondences are unknown



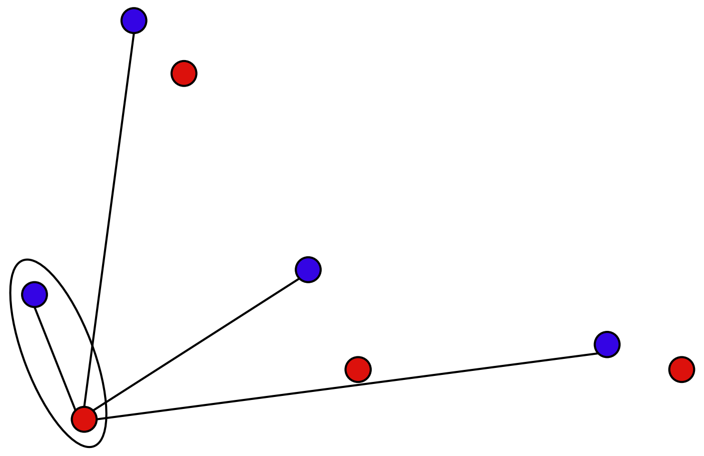
Iterative Closest Point Algorithm

- For each point, find the closest neighbor



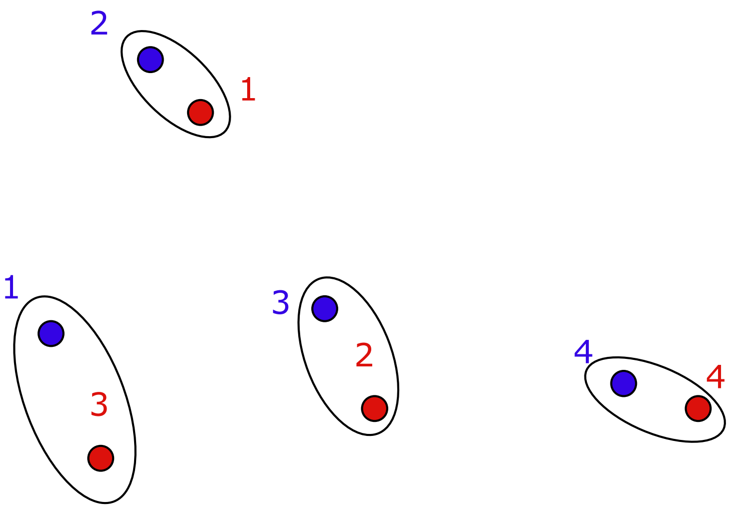
Iterative Closest Point Algorithm

- For each point, find the closest neighbor



Iterative Closest Point Algorithm

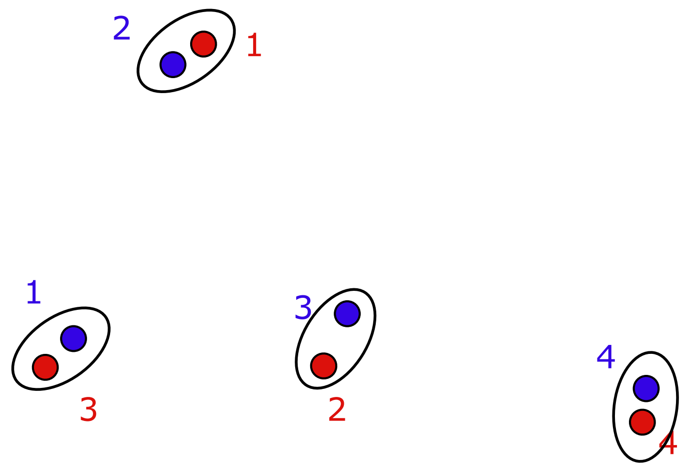
- This yields a list of point correspondences (point pairs)
- Estimate the transformation parameters on the basis of these correspondences



🟥	🟦	
1	2	$n(1) = 2$
2	3	$n(2) = 3$
3	1	$n(3) = 1$
4	4	$n(4) = 4$

Iterative Closest Point Algorithm

- Transform all points
- After the transformation:
  - Find new correspondences
  - Transform again…
- Until the transformation does not change



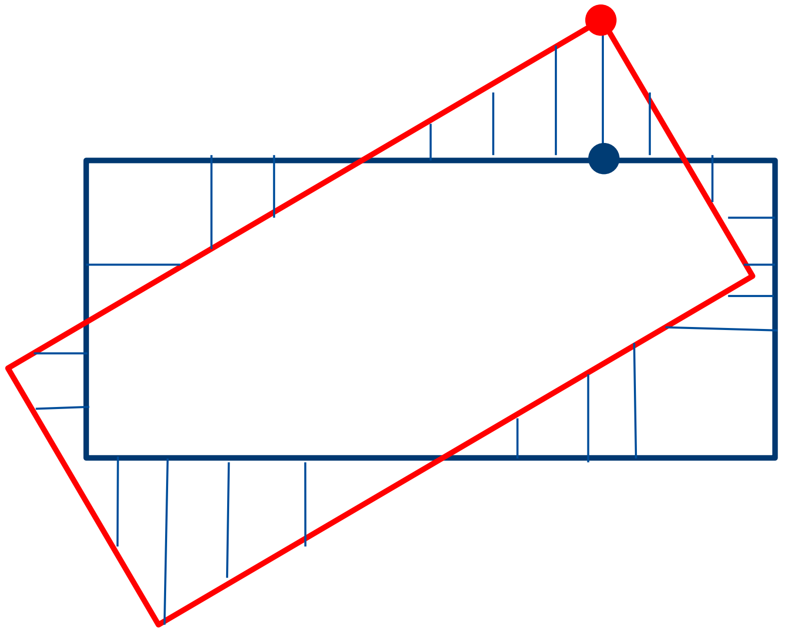
🟥	🟦
1	2
2	3
3	1
4	4

Here: correspondences have not changed → Stop

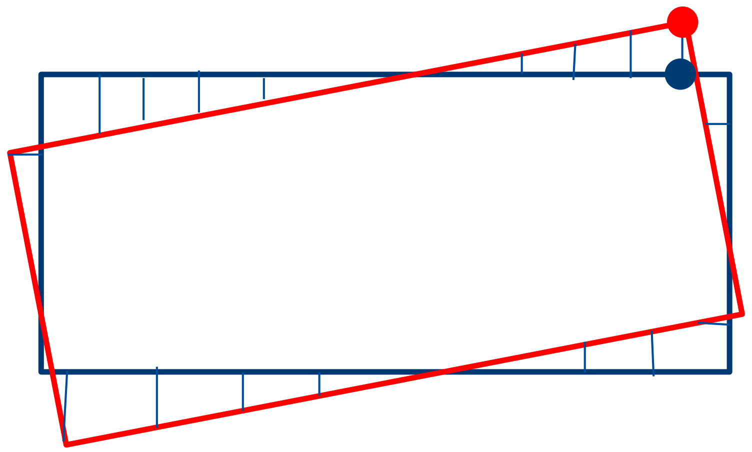
Iterative Closest Point Algorithm

- In this example: “correct” correspondences are found in the first iteration; from thereon they do not change
- In general:
  - It is accepted that initial assignments are wrong, but assumed (hoped for) that they improve with each iteration
  - There are no “correct” assignments usually. Even if two point clouds are aligned with each other, they usually do not contain identical object points.

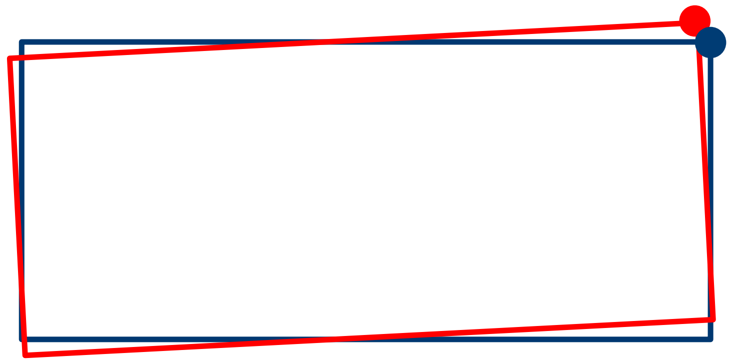
Example for ICP with changing correspondences



Example for ICP with changing correspondences



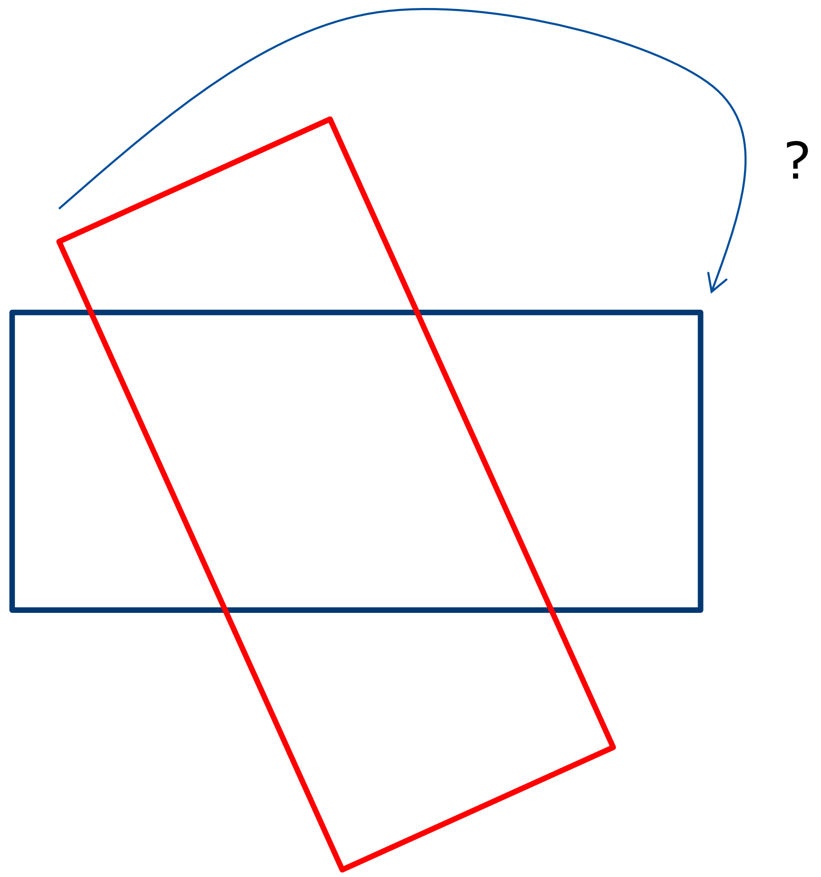
Example for ICP with changing correspondences



Example for ICP with changing correspondences

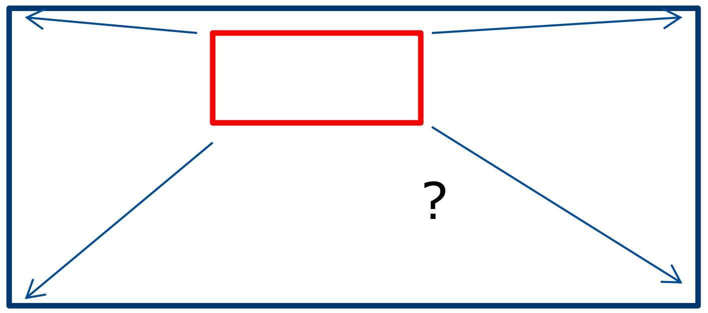
Example for ICP with changing correspondences

- Unlikely that the correct transformation will be found



Example for ICP with changing correspondences

- Unlikely that the correct transformation will be found



Convergence of the ICP algorithm

- The ICP algorithm **always converges**
- Although, as seen on the previous slides, not necessarily to a global minimum
- Remember the algorithm iterates the following two steps:
  1. For each point $\mathbf{l}_i$ in the left point list, find the closest point $\mathbf{r}_{n(i)}$ in the right point list
  2. Using these correspondences, re-estimate the transformation
- Consider the error (to minimize) in each step k→k+1 of the algorithm (parameters of the transform: $s^k, \mathbf{R}^k, \mathbf{t}^k$)

Convergence of the ICP algorithm

- Step 1: Re-assignment of the correspondences
- The total error of the previous iteration is:

$$e_k = \sum_i \left\| \left(s^k \mathbf{R}^k \mathbf{l}_i + \mathbf{t}^k \right) - \mathbf{r}_{n_k(i)} \right\|^2$$

- Now, new correspondences are established

$$\mathbf{l}_i \leftrightarrow \mathbf{r}_{n_{k+1}(i)}$$

  There are two cases:

  - A closer point is found. Then, the total error will decrease
  - The previously found assignment is unchanged. Then, the total error will remain unchanged, too

- Overall:

$$e'_k = \sum_i \left\| \left(s^k \mathbf{R}^k \mathbf{l}_i + \mathbf{t}^k \right) - \mathbf{r}_{n_{k+1}(i)} \right\|^2 \le e_k$$

Convergence of the ICP algorithm

- Step 2: Re-estimation of the transform
- The estimation of the transform finds the global minimum (when correspondences are given)
- Thus, the error is smaller or equal to the current error (which was given by the old transformation and the new assignments)

$$e_{k+1} = \sum_i \left\| \left(s^{k+1} \mathbf{R}^{k+1} \mathbf{l}_i + \mathbf{t}^{k+1} \right) - \mathbf{r}_{n_{k+1}(i)} \right\|^2 \leq e'_k$$

Convergence of the ICP algorithm

- Overall, for each step, the following holds:

$$e_{k+1} \leq e'_k \leq e_k$$

- Since $e_k \geq 0$ the sequence $\{e_k\}$ converges

- Reminder:
  - The estimation of the similarity transform yields the global minimum BUT
  - Overall, the ICP may end in a local minimum
  - This is due to the iterative point assignment, not the estimation of the transformation.

ICP Variants

ICP Variants

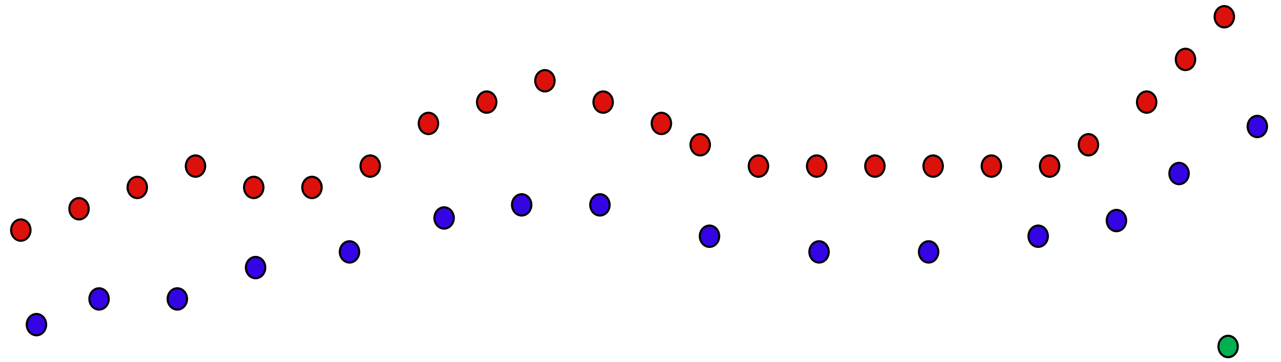
- Which aspects of the algorithm can be modified?
    1. Selection of points in one or in both datasets
    2. Assignment of points to elements in the other dataset
    3. Weighting of the correspondences
    4. Rejection of correspondences
    5. Error function and its minimization

Variants: selection of points

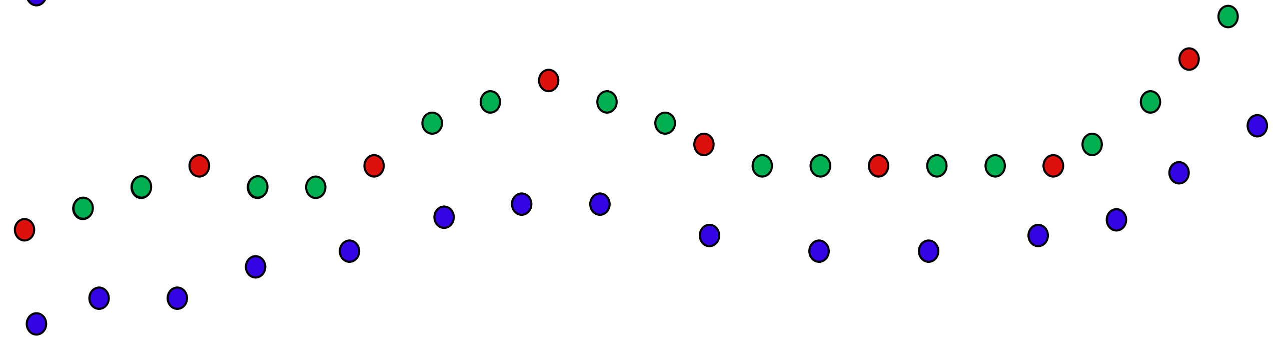
1. Take all points
2. Uniform draw of points
3. Uniform draw of points, in each iteration
4. Select points such that the distribution of normal vectors is homogeneous
5. If images are available: Select points which have a large (absolute) gradient in the image
6. All variants may select points in one dataset (and assign them to the full set in the other dataset), or select points in both datasets.

Variants: selection of points

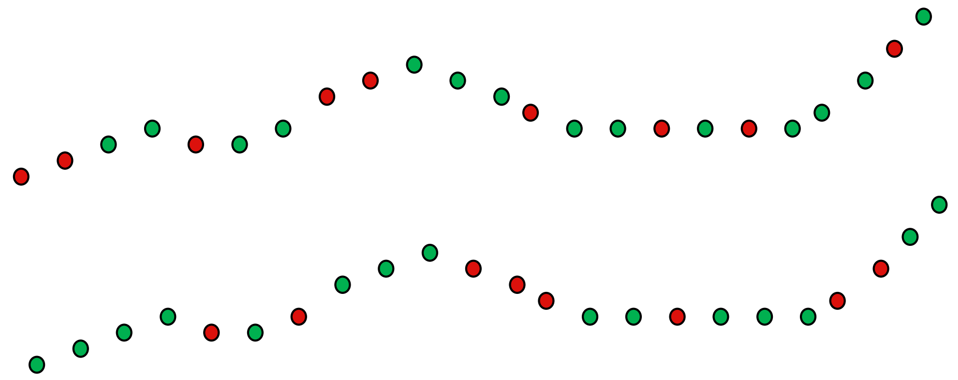
Take all



Uniform draw

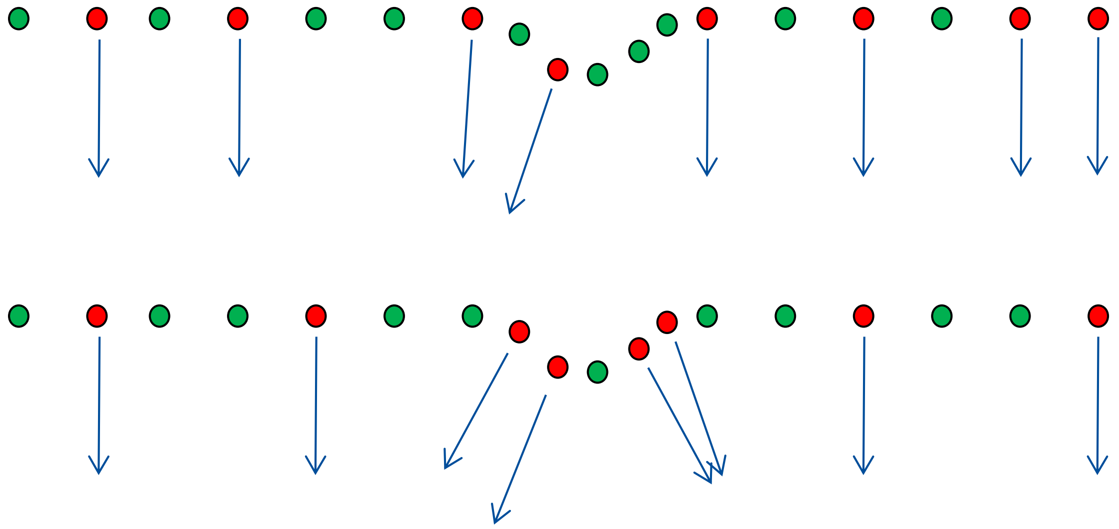


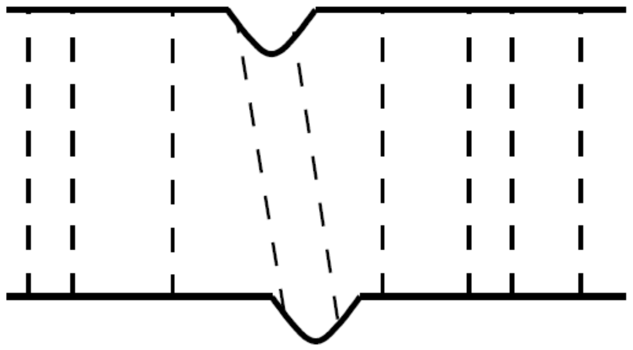
Uniform draw,
in each iteration



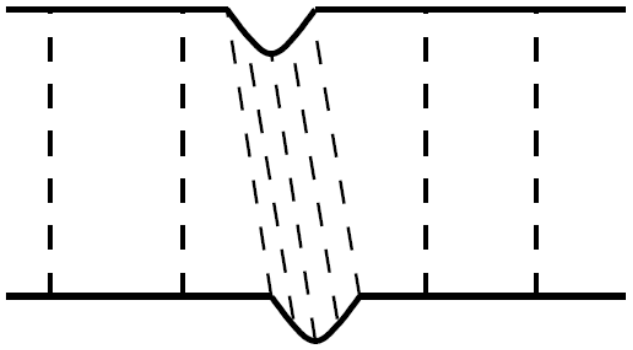
Variants: selection of points

Uniform distribution
of normal vectors





Random selection



Uniform distribution of normal vectors

Source: Rusinkiewicz

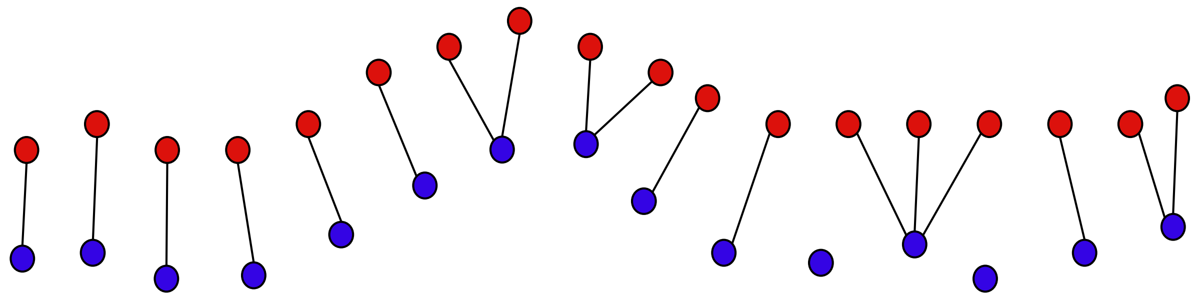
Variants: assignment of points

For each point **I** in the first list, to be assigned:

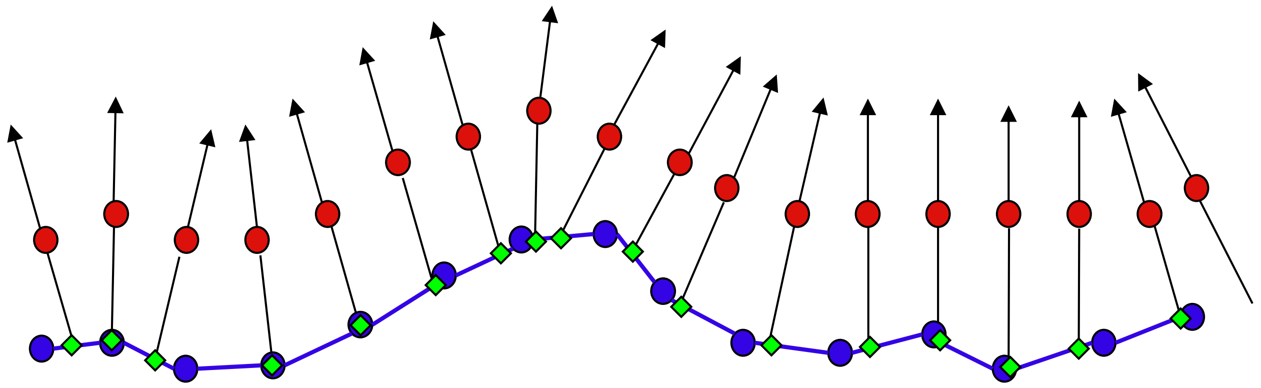
- Find the closest point in the other list
- Intersect the ray, which is defined by the point **I** and its normal vector, with the other surface („normal shooting")
- Project **I** onto the other surface, in the direction of capture of the other surface („reverse calibration")
- Project **I** to the other surface, then search on the surface for a good correspondence (point-to-point distance, matching of gray values or colors, etc.)
- Any of those methods, where only correspondences are accepted which have similar gray/color values, or similar normal vectors

Point assignment

- Closest point in the other dataset

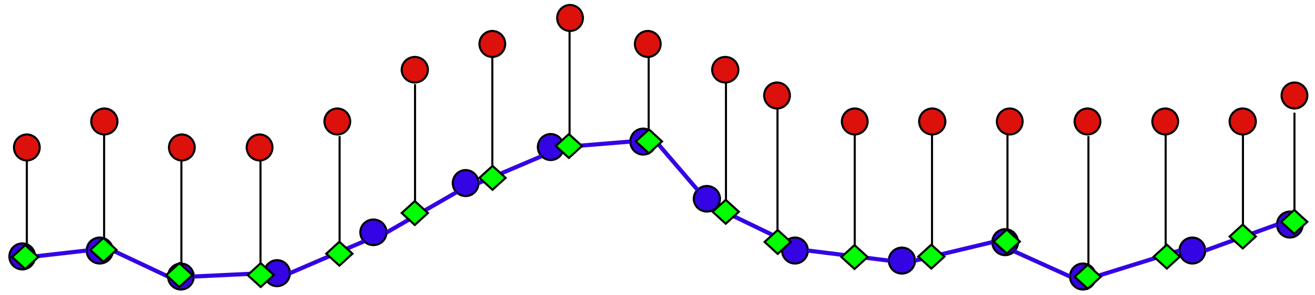


- Normal Shooting

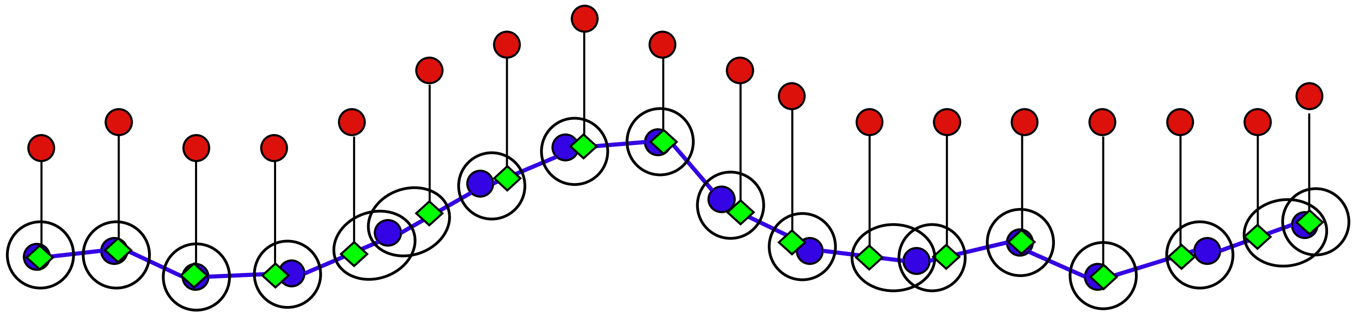


Point assignment

- Projection

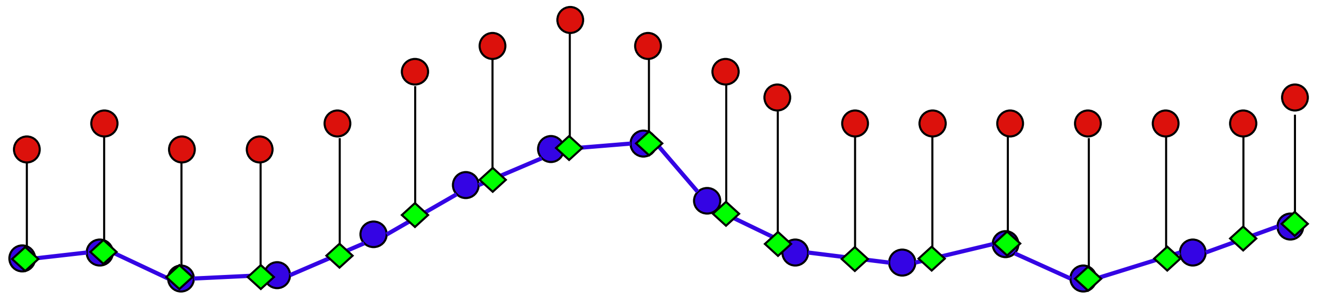


- Projection and search

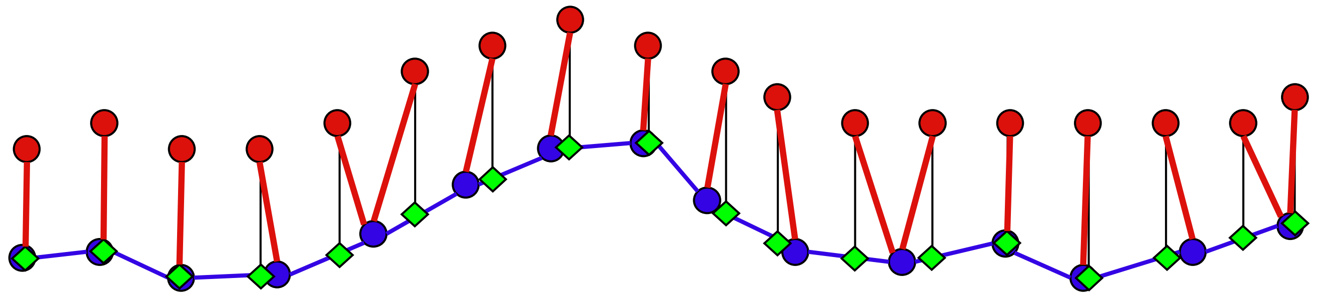


Point assignment

- Projection



- Projection and search



Weighting of correspondences

- Constant weight (1)
- Smaller weight for point pairs with a larger distance, e.g.

$$w = e^{-\frac{1}{2}\left(\frac{\mathrm{dist}(\mathbf{l}_i, \mathbf{r}_{n(i)})}{\sigma}\right)^2}$$

- Weight based on the similarity of the normal vectors

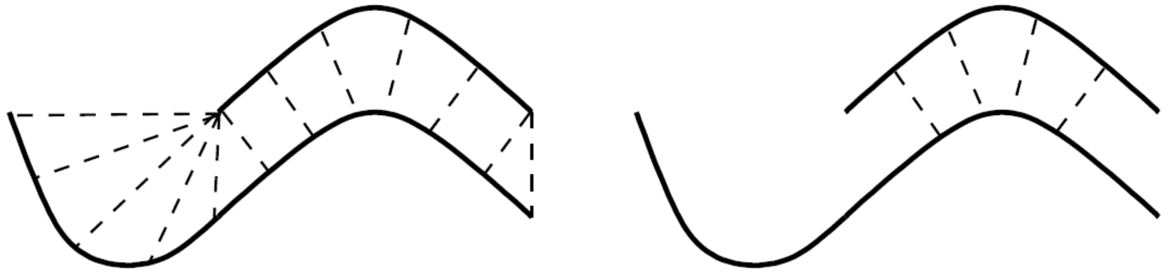
$$w = \left\langle \mathbf{n}_i , \mathbf{n}_{n(i)} \right\rangle = \cos(\angle(\mathbf{n}_i , \mathbf{n}_{n(i)}))$$

(Note: distinguish “front side” and “back side” of the surface)

Rejection of point correspondences

Reject…

- Points which are too far away (user-defined threshold)
- The p% worst correspondences (e.g. p% with largest distances)
- Points which have a distance of more than 2,5 x the standard deviation
- Pairs which are not consistent with neighboring pairs
- Pairs which include points along the edges of the datasets



Source: Rusinkiewicz

Error function and minimization

- Minimize the least squares distance
  - … with added gray value/color distance
- Use least squares distances from the source points to planes in the target (point-to-plane)
  - No closed form solution available
- "Least squares" = minimization of l_2-norm
  - Sensitive to outliers
  - Other (more robust, but also more costly) options:
    - l_1 norm, least absolute errors (deviations), solve by IRLS or can be transformed to a LP (linear programming) problem
    - maximum consensus, maximize the number of point pairs for which $\mathrm{dist}\left(\mathbf{l}_i, \mathbf{r}_{n(i)}\right) < \varepsilon$, leads to a combinatorial optimization problem, usually solved by search

Final notes

- For ICP, the initial transformation is important
  - E.g. try out different random initializations for the transformation
- Shown here: two datasets. General case: adjustment of *m* datasets
- Often, it is hard to determine if the estimated transformation is correct, since the “correct overlap” of two datasets is unknown.

References

- General: Manual of Photogrammetry, 5th ed.
- Solution to determine Euclidean transformation using quaternions was published by Sansò (1973) and later by B.K.P. Horn (1987). Here, we used the latter:
  - B.K.P. Horn, Closed-form solution of absolute orientation using unit quaternions, Journal of the Optical Society of America, vol. 4, pp 629, April 1987.
- ICP algorithm
  - Chen, Medioni, Object Modeling by Registration of Multiple Range Images, Proc. IEEE Conf. on Robotics and Automation, 1991
  - Besl, McKay, A Method for Registration of 3-D Shapes, Trans. PAMI, Vol. 14, No. 2, 1992
- Comparison of different ICP variants:
  - Rusinkiewicz, Levoy, Efficient Variants of the ICP Algorithm, International Conference on 3D Digital Imaging and Modeling (3DIM), 2001.