Registration

Compute the best transformation between two point clouds

Unstructured collections of points in N dimensions

Cases:

- You know which point in P corresponds to which in Q exactly
- You know correspondence, but with errors
- You don’t know correspondence

Application: building image mosaics

Find interest points in image A and image B

Build correspondences:

For each a in A find best matching b in B using descriptor

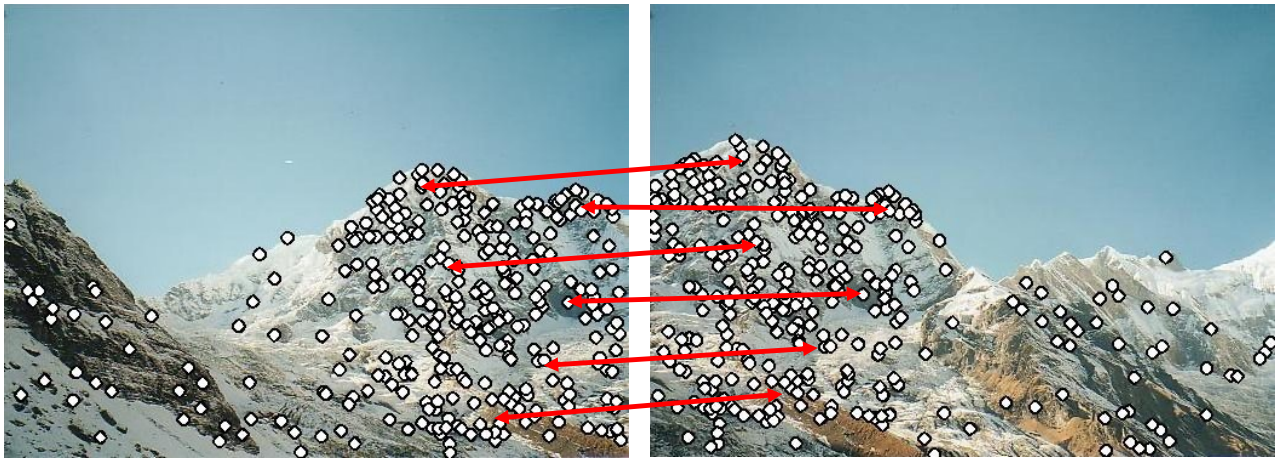
For each b in B find best matching a in A using descriptor

For consistent pairs, if descriptors are sufficiently similar

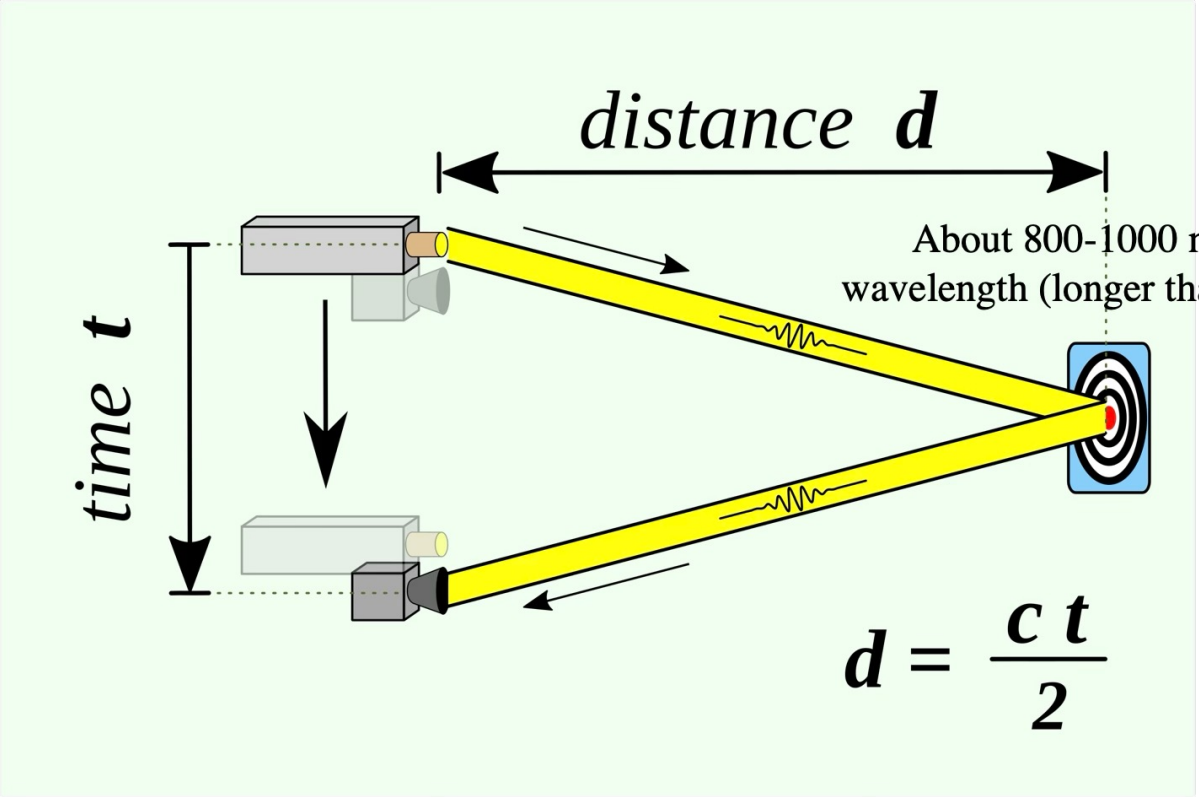
declare correspondence

Notice: you should get many correspondences BUT some are wrong

Recall…



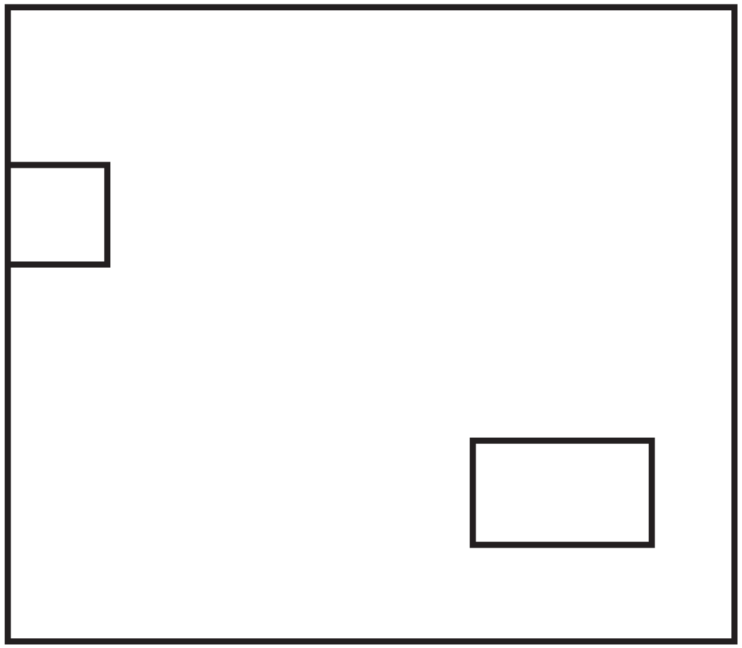
LIDAR produces point clouds



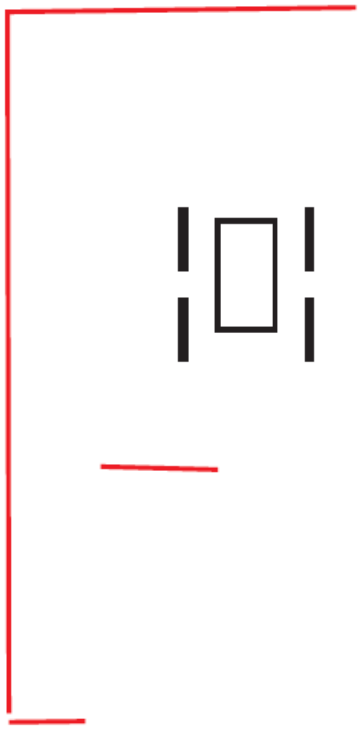
Wikipedia

LIDAR observations registered to map

Map

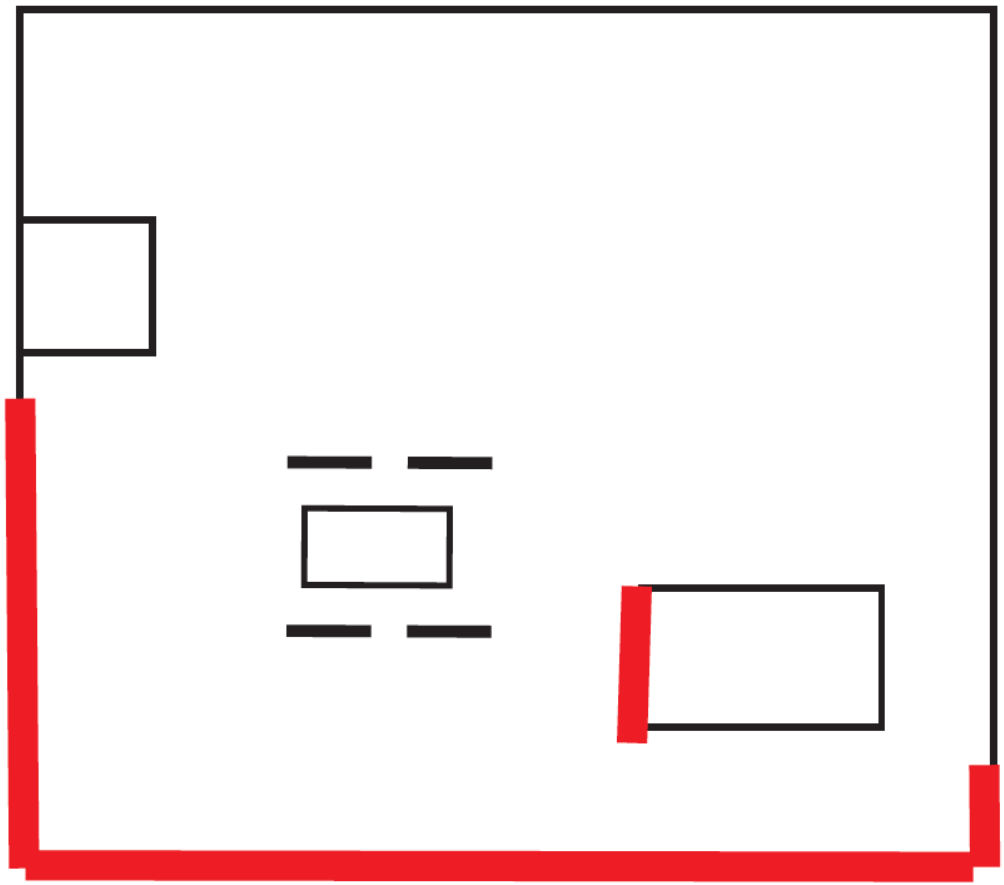


LIDAR



Yield your location

Map



Registering meshes to LIDAR

I have a CAD model of a car
   (large triangular mesh)

Where is this car in LIDAR data?
   registration problem

How is a mesh a point cloud?
   sample points on the mesh
   vertices

General remarks

Very like line fitting and line fitting recipes apply

The objects we are working with are now corresponding pairs
(point in A, point in B)

Outliers are usually correspondences that are wrong
there could be lots

Weighted least squares, affine tx, known csp

15.2.1 Affine Transformations

For an affine transformation, $\mathcal{T}(\mathbf{y})$ is $\mathcal{M}\mathbf{y}+\mathbf{t}$. Further, there is a transformation $\mathcal{T}$ so that $\mathcal{T}(\mathbf{y}_i)$ is close to $\mathbf{x}_i$ for each i. Write $\mathbf{r}_i$ for the vector from the transformed $\mathbf{y}_i$ to $\mathbf{x}_i$, so

$$\mathbf{r}_i(\mathcal{M},\mathbf{t}) = (\mathbf{x}_i - (\mathcal{M}\mathbf{y}_i + \mathbf{t}))$$

and

$$C_u(\mathcal{M},\mathbf{t}) = (1/N)\sum_i \mathbf{r}_i^T\mathbf{r}_i$$

should be small. Because it will be useful later, assume that there is a weight w_i for each pair and work with

$$C(\mathcal{M},\mathbf{t}) = \sum_i w_i\mathbf{r}_i^T\mathbf{r}_i$$

where $w_i = 1/N$ if points all have the same weight. The gradient of this cost with

Solving for translation

$$\mathbf{r}_i(\mathcal{M}, \mathbf{t}) = (\mathbf{x}_i - (\mathcal{M}\mathbf{y}_i + \mathbf{t}))$$

$$C(\mathcal{M}, \mathbf{t}) = \sum_i w_i \mathbf{r}_i^T \mathbf{r}_i$$

where $w_i = 1/N$ if points all have the same weight. The gradient of this cost with respect to $\mathbf{t}$ is

$$-2 \sum_i w_i \left(\mathbf{x}_i - \mathcal{M}\mathbf{y}_i - \mathbf{t}\right)$$

which vanishes at the solution, so that

$$\mathbf{t} = \frac{\sum_i w_i \mathbf{x}_i - \mathcal{M} \sum w_i \mathbf{y}_i}{\sum_i w_i}.$$

Solving for translation

Now if $\sum_i w_i \mathbf{x}_i = \sum_i w_i \mathcal{M}\mathbf{y}_i = \mathcal{M}(\sum_i w_i \mathbf{y}_i)$, then $\mathbf{t} = \mathbf{0}$. An easy way to achieve $\mathbf{t} = \mathbf{0}$ is to ensure $\sum_i w_i \mathbf{x}_i = 0$ and $\sum_i w_i \mathbf{y}_i = 0$. Write

$$\mathbf{c}_x = \frac{\sum_i w_i \mathbf{x}_i}{\sum_i w_i}$$

for the center of gravity of the observations (etc.) Now form

$$\mathbf{u}_i = \mathbf{x}_i - \mathbf{c}_x \text{ and } \mathbf{v}_i = \mathbf{y}_i - \mathbf{c}_y$$

and if you use $\mathcal{U}$ and $\mathcal{V}$, then the translation will be zero and must only estimate $\mathcal{M}$. Further, the estimate $\hat{\mathcal{M}}$ of this matrix yields that the translation from the original reference points to the original observations is $\mathbf{c}_x - \hat{\mathcal{M}}\mathbf{c}_y$.

Finding M (compact form)

Finding $\mathcal{M}$ now reduces to minimizing

$$\sum_i w_i \left(\mathbf{u}_i - \mathcal{M}\mathbf{v}_i\right)^T \left(\mathbf{u}_i - \mathcal{M}\mathbf{v}_i\right)$$

as a function of $\mathcal{M}$. The natural procedure – take a derivative and set to zero, and obtain a linear system (**exercises**) – works fine, but it is helpful to apply some compact and decorative notation.

Finding M (long form)

Write $\mathcal{W} = \mathsf{diag}\left([w_1, \ldots, w_N]\right)$, $\mathcal{U} = \left[\mathbf{u}_1^T, \ldots, \mathbf{u}_N^T\right]$ (and so on). Recall all vectors are column vectors, so $\mathcal{U}$ is $N \times d$. You should check that the objective can be rewritten as

$$\mathsf{Tr}\left(\mathcal{W}(\mathcal{U} - \mathcal{V}\mathcal{M}^T)(\mathcal{U} - \mathcal{V}\mathcal{M}^T)^T\right).$$

exercises Now the trace is linear; $\mathcal{U}^T\mathcal{W}\mathcal{U}$ is constant;

$$\mathsf{Tr}\left(\mathcal{A}\right) = \mathsf{Tr}\left(\mathcal{A}^T\right);$$

and

$$\mathsf{Tr}\left(\mathcal{A}\mathcal{B}\mathcal{C}\right) = \mathsf{Tr}\left(\mathcal{B}\mathcal{C}\mathcal{A}\right) = \mathsf{Tr}\left(\mathcal{C}\mathcal{A}\mathcal{B}\right)$$

(check this by writing it out, and remember it; it's occasionally quite useful). This

Finding M (long form)

(check this by writing it out, and remember it; it's occasionally quite useful). This means the cost is equivalent to

$$\mathrm{Tr}\left(-2\mathcal{U}^T\mathcal{W}\mathcal{V}\mathcal{M}^T\right) + \mathrm{Tr}\left(\mathcal{M}\mathcal{V}^T\mathcal{W}\mathcal{V}\mathcal{M}^T\right)$$

which will be minimized when

$$\mathcal{M}\mathcal{V}^T\mathcal{W}\mathcal{V} = \mathcal{U}^T\mathcal{W}\mathcal{V}$$

(which you should check **exercises**). The exercises establish cases where $\mathcal{V}^T\mathcal{W}\mathcal{V}^T$ will have full rank, and in these – the usual – cases $\mathcal{M}$ is easily obtained **exercises** . Notice this derivation works *whatever* the dimension of the points.

Euclidean motion

Most interesting in 2D or 3D

The matrix is a rotation matrix

You can do this in closed form (not widely known)

Use centers of gravity, as above.

As in the previous section, subtract the centers of gravity to get the translation, and work with $\mathbf{u}_i$ and $\mathbf{v}_i$. The problem is now to choose $\mathcal{R}$ to minimize

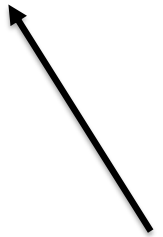
$$\sum_i w_i(\mathbf{u}_i - \mathcal{R}\mathbf{v}_i)^T(\mathbf{u}_i - \mathcal{R}\mathbf{v}_i).$$

Euclidean motion

$$\sum_i w_i(\mathbf{u}_i - \mathcal{R}\mathbf{v}_i)^T(\mathbf{u}_i - \mathcal{R}\mathbf{v}_i) = \mathrm{Tr}\left(\mathcal{W}(\mathcal{U} - \mathcal{V}\mathcal{R}^T)(\mathcal{U} - \mathcal{V}\mathcal{R})^T\right)$$

$$= \mathrm{Tr}\left(-2\mathcal{V}^T\mathcal{W}\mathcal{U}\mathcal{R}\right) + K$$

$$(\text{because } \mathcal{R}^T\mathcal{R} = \mathcal{I})$$



Terms not involving R, so of no interest

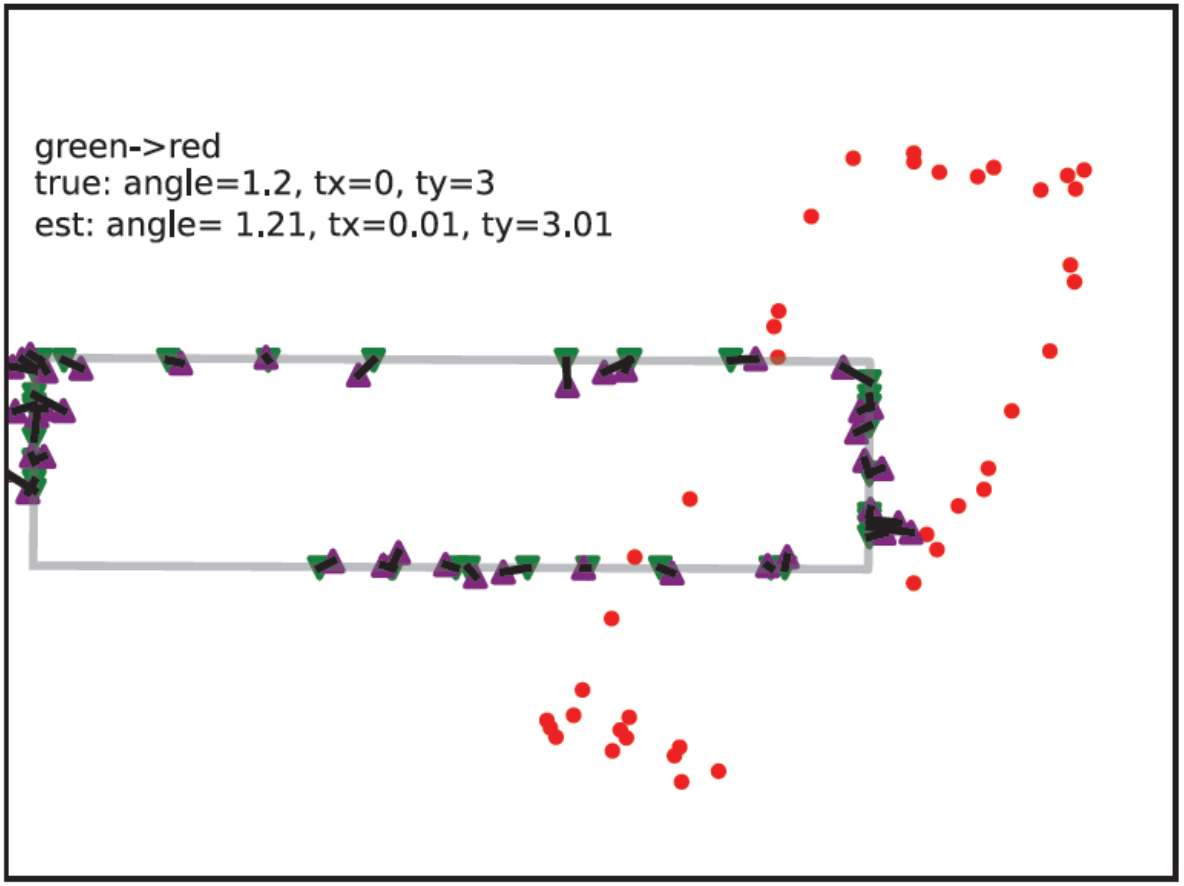
Euclidean motion

$$\mathsf{Tr}\left(-2\mathcal{V}^T\mathcal{W}\mathcal{U}\mathcal{R}\right)$$

Here K is a constant that doesn't involve $\mathcal{R}$ and so is of no interest. Now compute an SVD of $\mathcal{V}^T\mathcal{W}\mathcal{U}$ to obtain $\mathcal{V}^T\mathcal{W}\mathcal{U} = \mathcal{A}\Sigma\mathcal{B}^T$ where $\mathcal{A}$, $\mathcal{B}$ are orthonormal, and $\mathcal{S}$ is diagonal (Section 15.10 if you're not sure). Now $\mathcal{B}^T\mathcal{R}\mathcal{A}$ is orthonormal, and we must maximize $\mathsf{Tr}\left(\mathcal{B}^T\mathcal{R}\mathcal{A}\mathcal{S}\right)$, meaning $\mathcal{B}^T\mathcal{R}\mathcal{A} = \mathcal{I}$ (check this if you're not certain), and so $\mathcal{R} = \mathcal{B}\mathcal{A}^T$.

Euclidean est. well behaved under noise

0.05



Green triangles – target pts lying on gray rectangle

Red dots – source

(target points transformed, then noise added)

Purple triangles – apply estimated transformation to red points

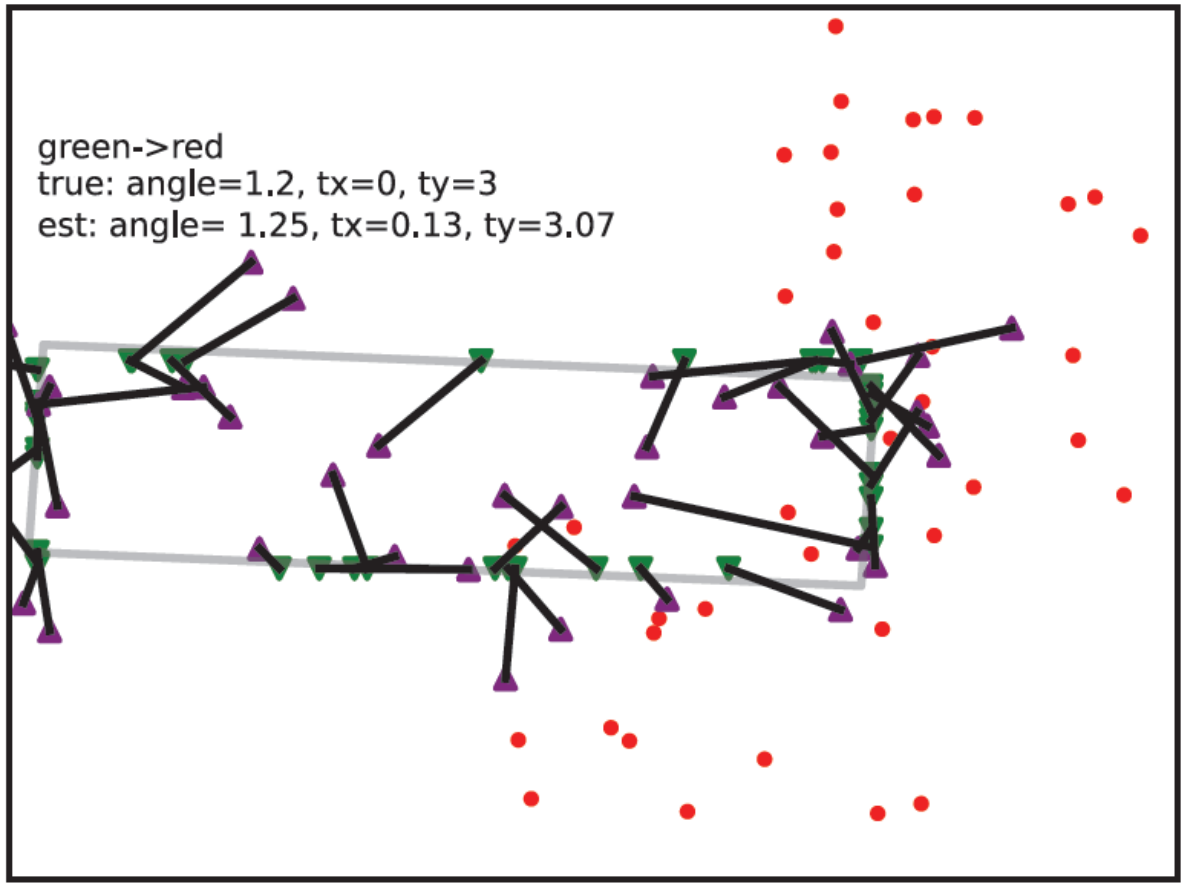
Gray rectangle – transformation applied to true rectangle underlying red points

Notice:

transformation is about right, not massively disrupted by noise

Euclidean est. well behaved under noise

0.3



Green triangles – target pts lying on gray rectangle

Red dots – source

(target points transformed, then noise added)

Purple triangles – apply estimated transformation to red points

Gray rectangle – transformation applied to true rectangle underlying red points

Notice:

transformation is about right, only somewhat disrupted by noise

Projective transformations – 2D

Recall from Section 3.2 that a projective transformation of an image is given by a 3×3 matrix $\mathcal{M}$ that has full rank. The transformation can be written

$$\begin{bmatrix} x_1 \\ \\ x_2 \end{bmatrix} = \begin{bmatrix} \dfrac{m_{11}y_1 + m_{12}y_2 + m_{13}}{m_{31}y_1 + m_{32}y_2 + m_{33}} \\ \\ \dfrac{m_{21}y_1 + m_{22}y_2 + m_{23}}{m_{31}y_1 + m_{32}y_y + m_{33}} \end{bmatrix}.$$

Projective transformations – dD

Higher dimensions follow the pattern. A projective transformation in d dimensions is given by a $d+1 \times d+1$ matrix $\mathcal{M}$ that has full rank. The transformation is now

$$\begin{bmatrix} x_1 \\ \dots \\ x_d \end{bmatrix} = \begin{bmatrix} \dfrac{m_{11}y_1 + \ldots + m_{1d}y_d + m_{1(d+1)}}{m_{(d+1)1}y_1 + \ldots + m_{(d+1)d}y_d + m_{(d+1)(d+1)}} \\ \dots \\ \dfrac{m_{d1}y_1 + \ldots + m_{dd}y_d + m_{d(d+1)}}{m_{(d+1)1}y_1 + \ldots + m_{(d+1)d}y_d + m_{(d+1)(d+1)}} \end{bmatrix}.$$

Projective transformations - solving

A weighted least squares solution now solves

$$\sum_i w_i \mathbf{r}_i^T \mathbf{r}_i.$$

There isn't a clean form for the solution, and numerical minimization is required. You should use a second order method (Levenberg-Marquardt is favored; Chapter 15.10). Experience teaches that this optimization is not well behaved without a strong start point.

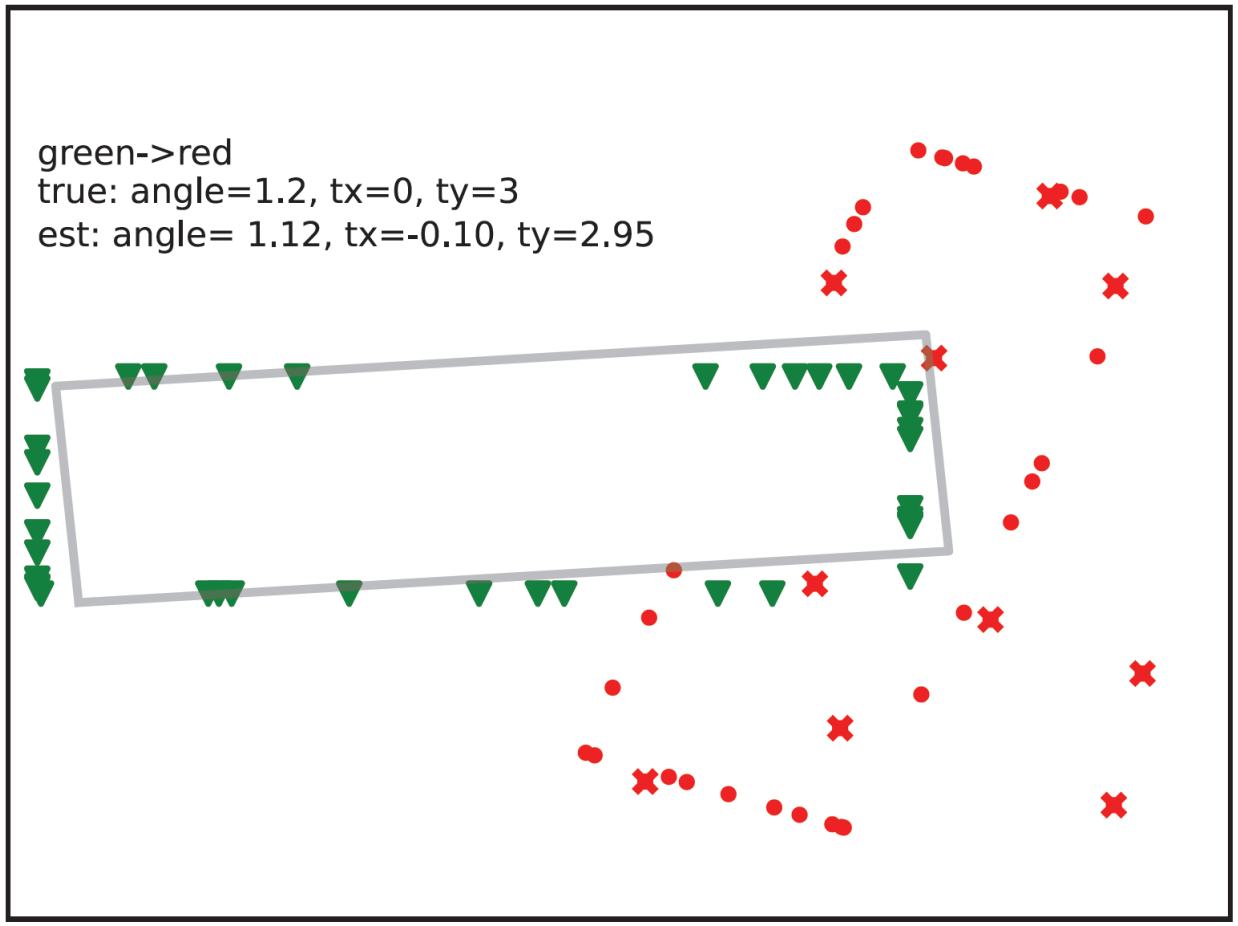
A start point

There is an easy construction for a good start point. For a pair of known points $\mathbf{x}$ and $\mathbf{y}$, you can cross multiply the equations for the projective transformation to get

$$\begin{bmatrix} 0 \\ \dots \\ 0 \end{bmatrix} = \begin{bmatrix} \left(m_{11}y_1 + \ldots + m_{1d}y_d + m_{1(d+1)}\right) - \\ x_1\left(m_{(d+1)1}y_1 + \ldots + m_{(d+1)d}y_d + m_{(d+1)(d+1)}\right) \\ \dots \\ \\ \left(m_{d1}y_1 + \ldots + m_{dd}y_d + m_{d(d+1)}\right) - \\ x_d\left(m_{(d+1)1}y_1 + \ldots + m_{(d+1)d}y_d + m_{(d+1)(d+1)}\right) \end{bmatrix}.$$

Here the m_{ij} are unknown, so this is a set of d homogenous linear equations in $(d+1) \times (d+1)$ unknowns. In turn, if you have at least $d+1$ different $(\mathbf{x}, \mathbf{y})$ pairs that meet conditions **exercises** , you can solve the system up to scale. But the scale of the solution does not affect the transformation it implements, so you have a start point. The resulting estimate of $\mathcal{M}$ has a good reputation as a start point for a full optimization. Notice this construction does not take weights into account. If the weights come from IRLS, then you need this construction only at the start. For every other iteration, the previous iteration will supply an acceptable start point as well as weights.

Outliers



Green triangles – target pts lying on gray rectangle

Red dots – source

(target points transformed, then noise added)

Red x – outliers on source

Gray rectangle – transformation applied to true rectangle underlying red points

Notice:

transformation is disrupted by outliers

IRLS

Start with initial transformation

get weights, scale from transformation

Iterate:

estimate transformation using weights, scale

estimate scale using transformation

estimate weights using scale, transformation

We *know* that one stationary point is the true minimum

No other guarantees I'm aware of, but quite well behaved

IRLS applies

The IRLS recipe can be applied with very little modification to registration. Choose a robust cost function from Section 13.2.1 or elsewhere. Recall this cost applies to the residual. Write θ for the parameters of the transformation $\mathcal{T}_\theta$, and the residual is now

$$r(\mathbf{x}_i, \mathbf{y}_i, \theta) = \sqrt{(\mathbf{x}_i - \mathcal{T}_\theta(\mathbf{y}_i))^T(\mathbf{x}_i - \mathcal{T}_\theta(\mathbf{y}_i))}.$$

The square root ensures that minimizing the least squares criterion is equivalent to

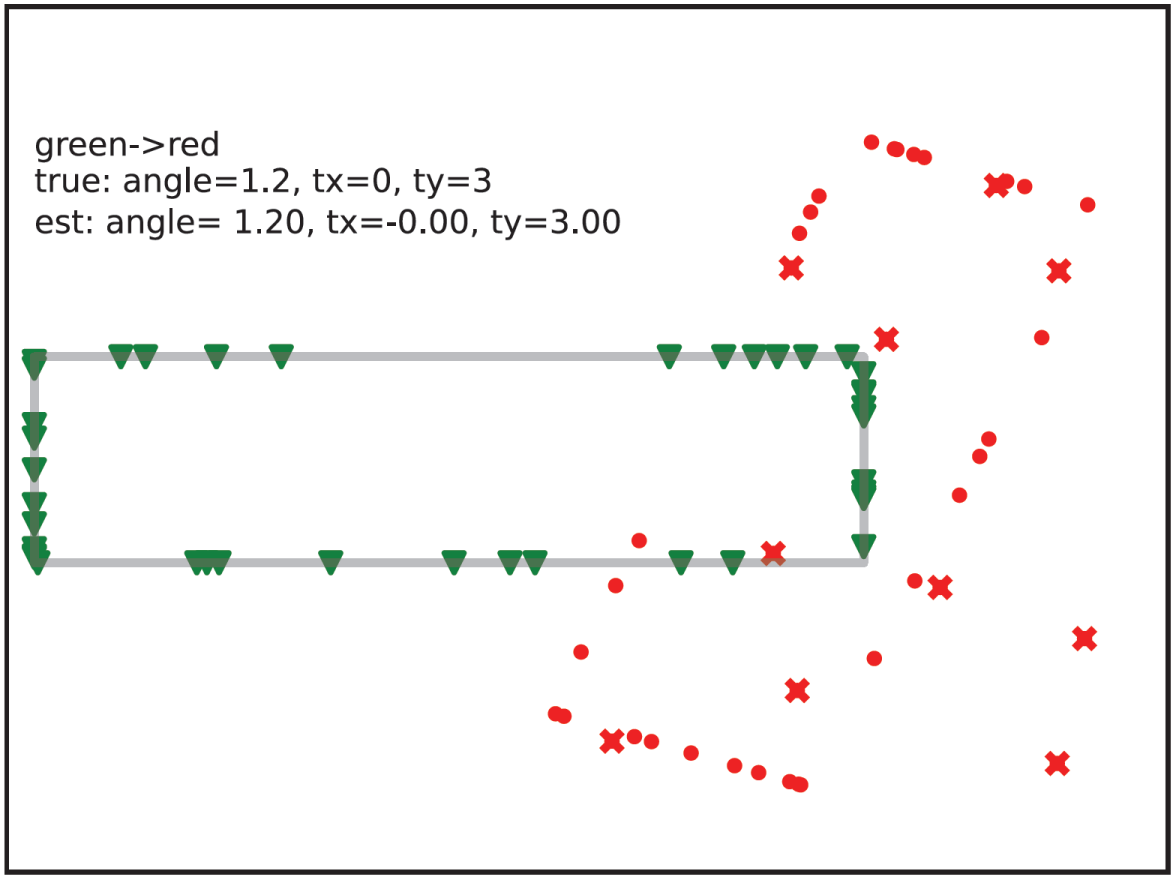
$$(1/2)\sum_i (r(\mathbf{x}_i, \mathbf{y}_i, \theta))^2.$$

For any given θ, the weights are now

$$w_i = \left(\frac{\frac{\partial \rho}{\partial u}}{r(\mathbf{x}_i, \mathbf{y}_i, \theta)} \right).$$

Outliers

IRLS, 5 outliers



Green triangles – target pts lying on gray rectangle

Red dots – source

(target points transformed, then noise added)

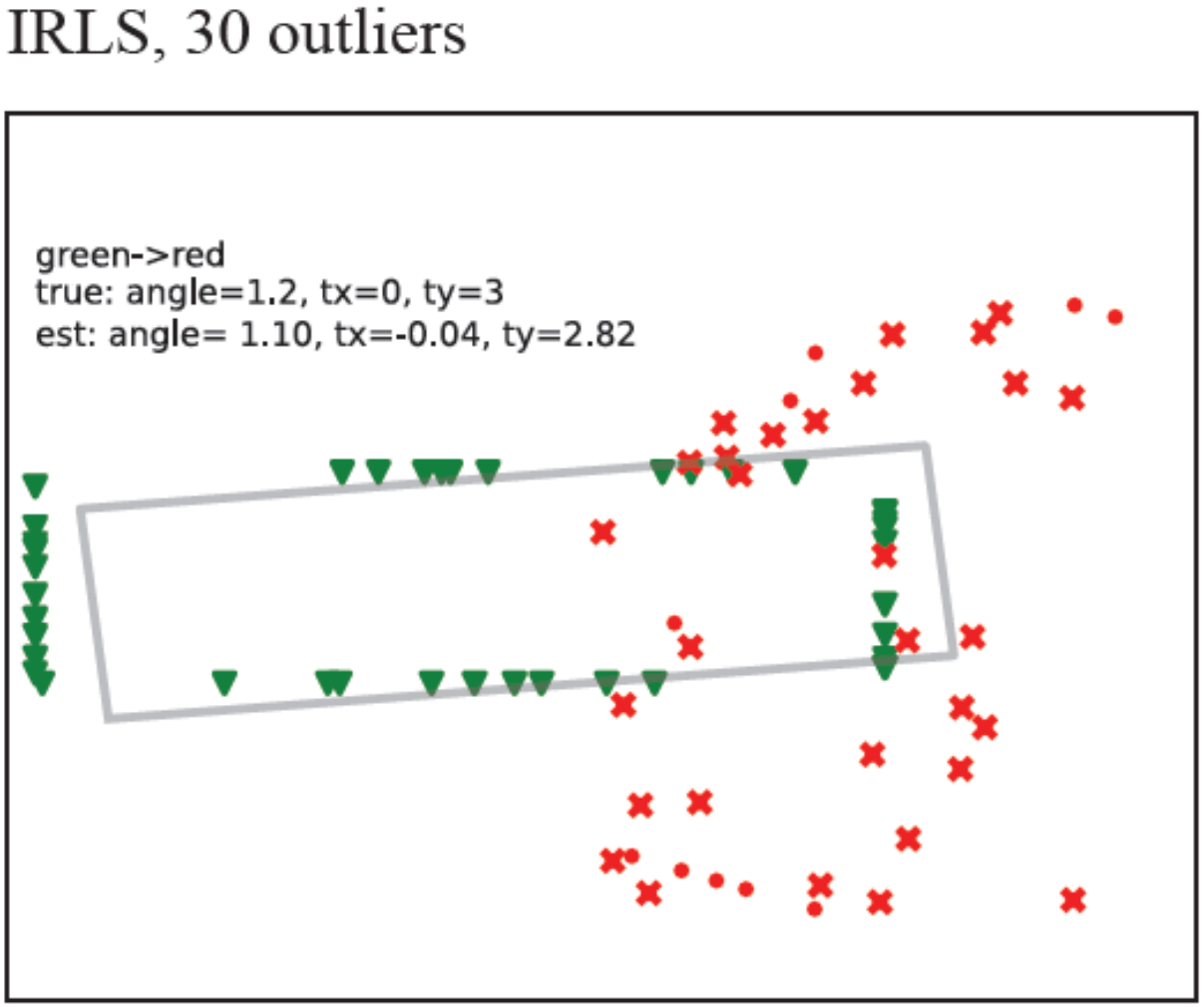
Red x – outliers on source

Gray rectangle – transformation applied to true rectangle underlying red points

Notice:

transformation is NOT disrupted by outliers

IRLS breaks down when there are too many outliers



Green triangles – target pts lying on gray rectangle

Red dots – source

(target points transformed, then noise added)

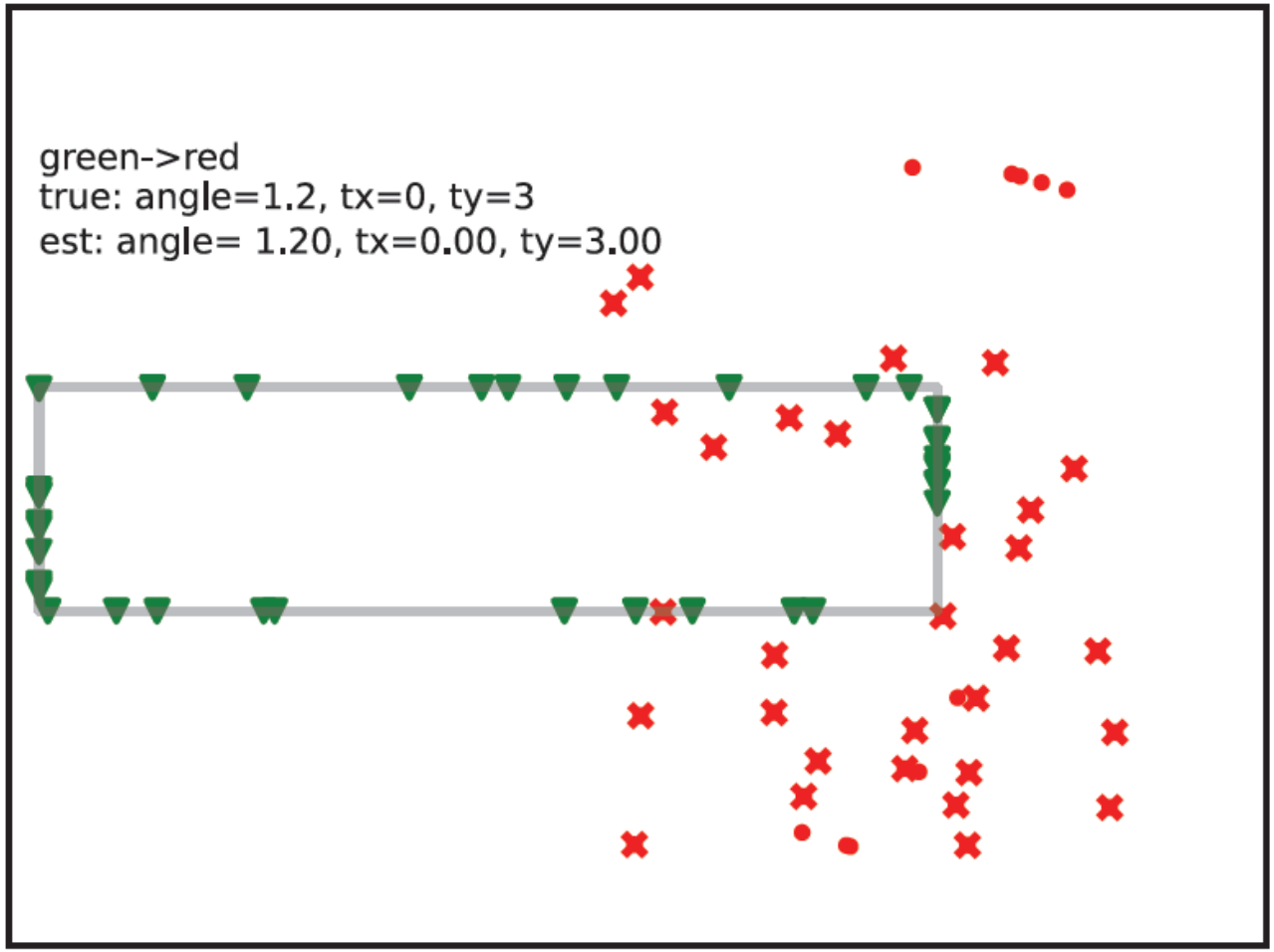
Red x – outliers on source

Gray rectangle – transformation applied to true rectangle underlying red points

Notice:

transformation IS disrupted by outliers

RANSAC to the rescue



Green triangles – target pts lying on gray rectangle

Red dots – source

(target points transformed, then noise added)

Red x – outliers on source

Gray rectangle – transformation applied to true rectangle underlying red points

Notice:

transformation IS disrupted by outliers

RANSAC

Affine transformation: d+1 correspondences in d dim

Projective transformation: d+2 correspondences in d dim

Euclidean:

- use 2 for plane (2D)
- use 3 for 3D

BUT some such are obvious outliers

Key Issue: there can be a lot of outliers

What if you don’t know correspondences?

RANSAC isn’t usually enough

Registration produces very large quantities of outliers

Image A has N points

Image B has M points

Idea:

use every pair, and accept you have outliers -- this is unwise!

Problem:

M N pairs

but only at most min(M, N) are good

A very important reason to care about interest points and descriptors

Iterated closest points or ICP

Idea:

If the transformation is nearly the identity,

then nearest point is likely correspondence

Strategy:

Start with good transformation

Iterate:

Estimate correspondences assuming tx is right

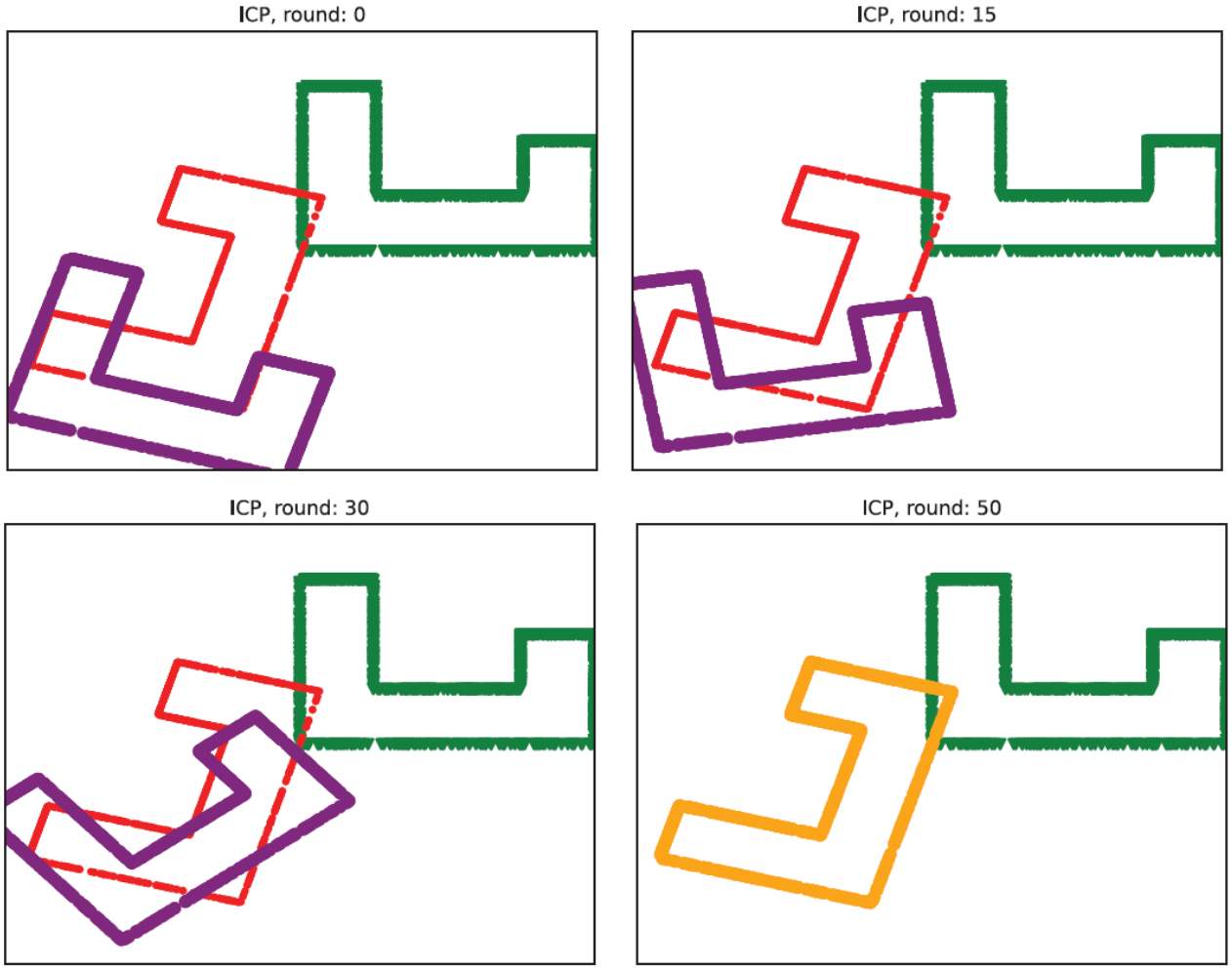
Re-estimate tx

ICP

Formally, start with a transformation estimate $\mathcal{T}_1$, a set of $\mathbf{m}_i^{(1)} = \mathcal{T}^{(1)}(\mathbf{y}_i)$ (the *running points*) and then repeat three steps:

- **Estimate correspondences** using the transformation estimate. Then, for each $\mathbf{x}_i$, we find the closest $\mathbf{m}^{(n)}$ (say $\mathbf{m}_c^{(n)}$); then $\mathbf{x}_i$ corresponds to $\mathbf{m}_{c(i)}^{(n)}$.
- **Estimate a transformation** $\mathcal{T}^{(n+1)}$ using the corresponding pairs.
- **Update the running points** by mapping $\mathbf{m}_i^{(n)}$ to $\mathcal{T}^{(n+1)}(\mathbf{m}_i^{(n)})$ and

Can converge quite fast



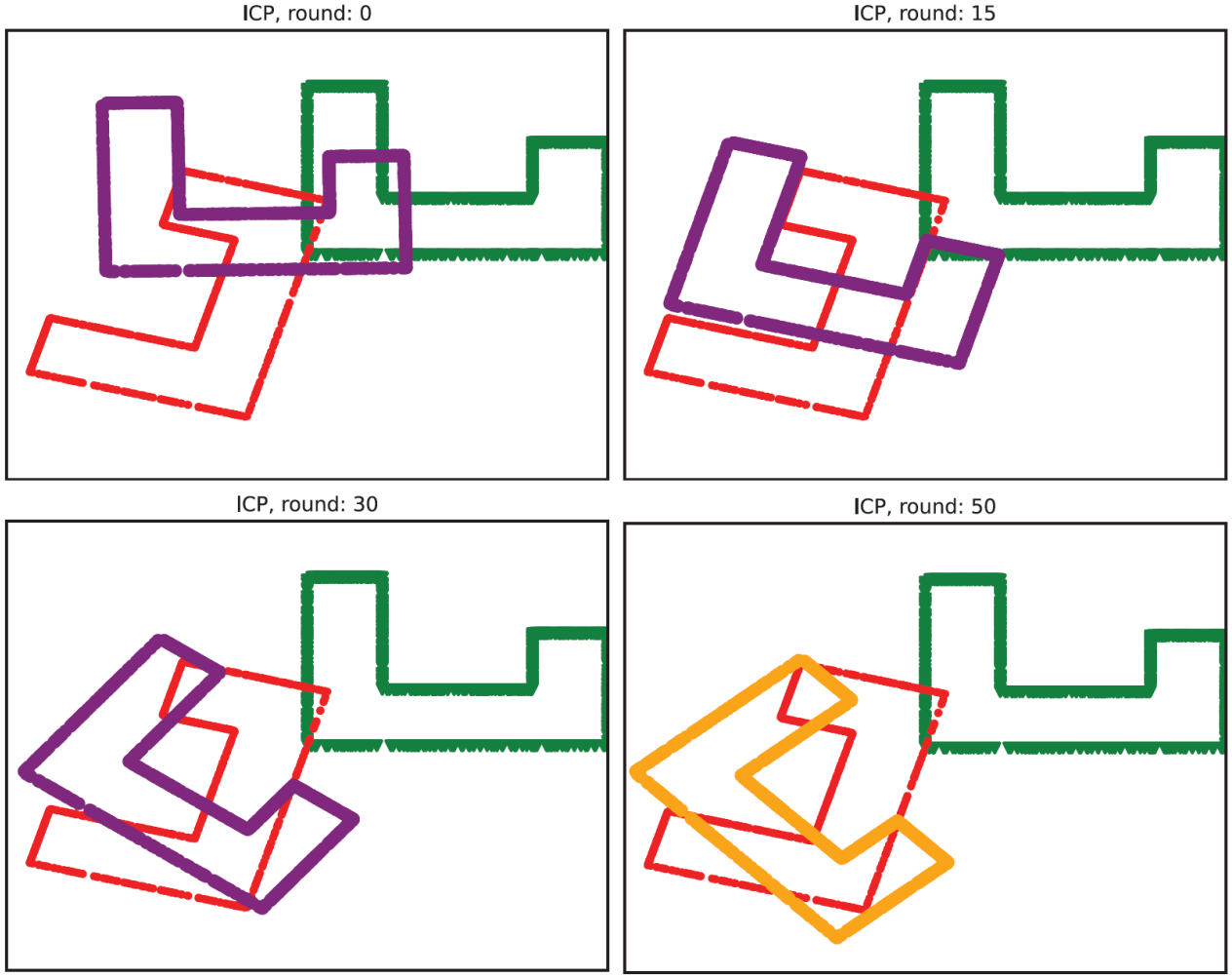
Red – target

Green – source

Purple – running points

Green at 0 -> Purple at 0 – initial transformation

ICP doesn't always converge



Red – target

Green – source

Purple – running points

Green at 0 -> Purple at 0 – initial transformation

ICP Issues

You have to do an awful lot of nearest neighbors

- particularly if the point cloud is big
- subsample the point cloud
- approximate nearest neighbors

There are still robustness issues

Ideas:

- drop correspondences for large distances
- use IRLS at each round

Resampling

If the two point clouds are big, resample

- Choose some number N of points that is acceptable
- Draw N points uniformly and at random from point clouds
  - Do this with replacement, because its easier

Approximate nearest neighbors

ISSUE:

do you build a new tree for every iteration?

Strategies:

Space is almost always 2D or 3D, so you can grid it

It is easy to test what grid bin a point falls in (truncate, round)

Build a big enough grid and use that

ICP Issues

You have to do an awful lot of nearest neighbors

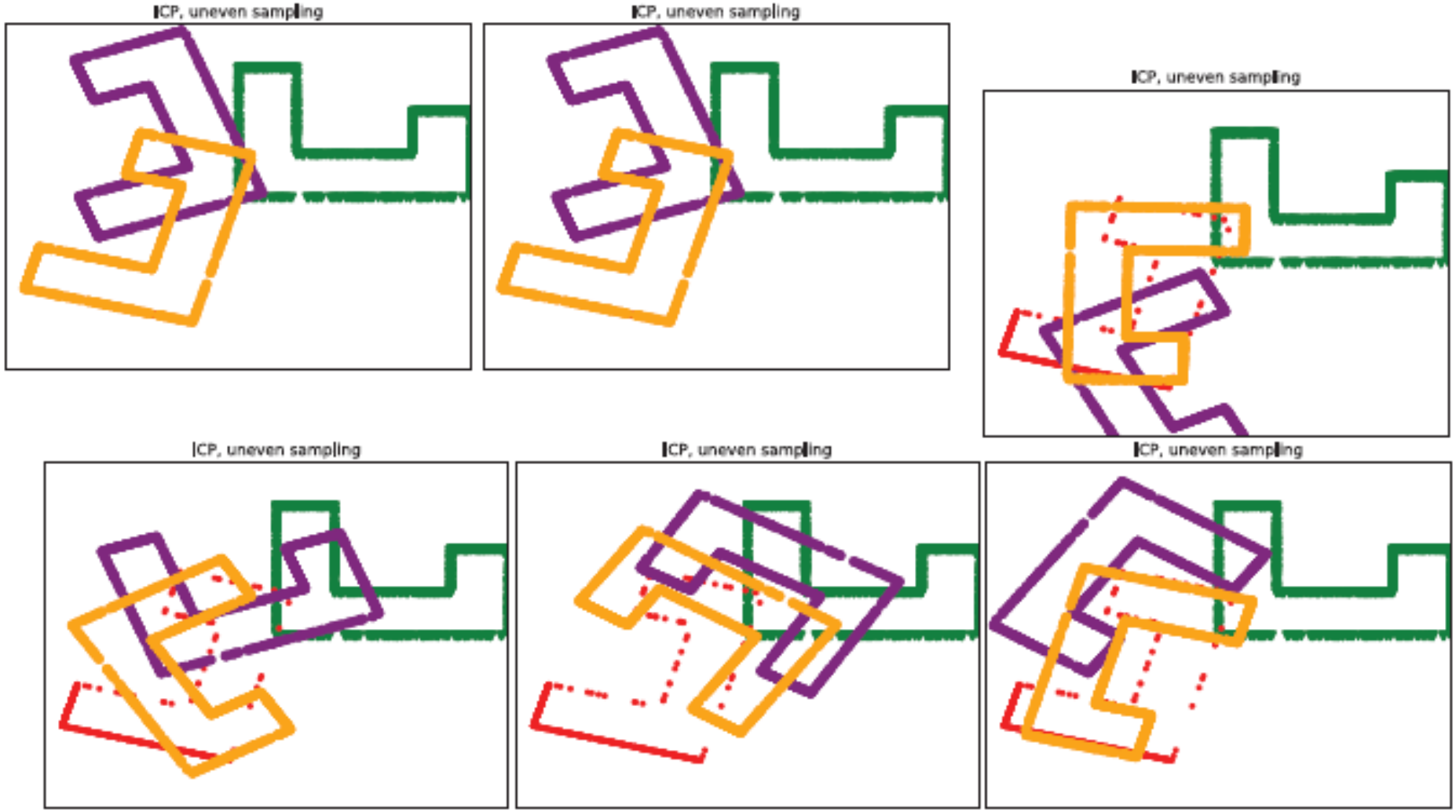
- particularly if the point cloud is big
- subsample the point cloud
- approximate nearest neighbors

There are still robustness issues

Ideas:

- drop correspondences for large distances
- use IRLS at each round

Uneven sampling creates ICP problems



Stratifying a sample

Cut the space into even bins

for example, a grid

Read points into bins

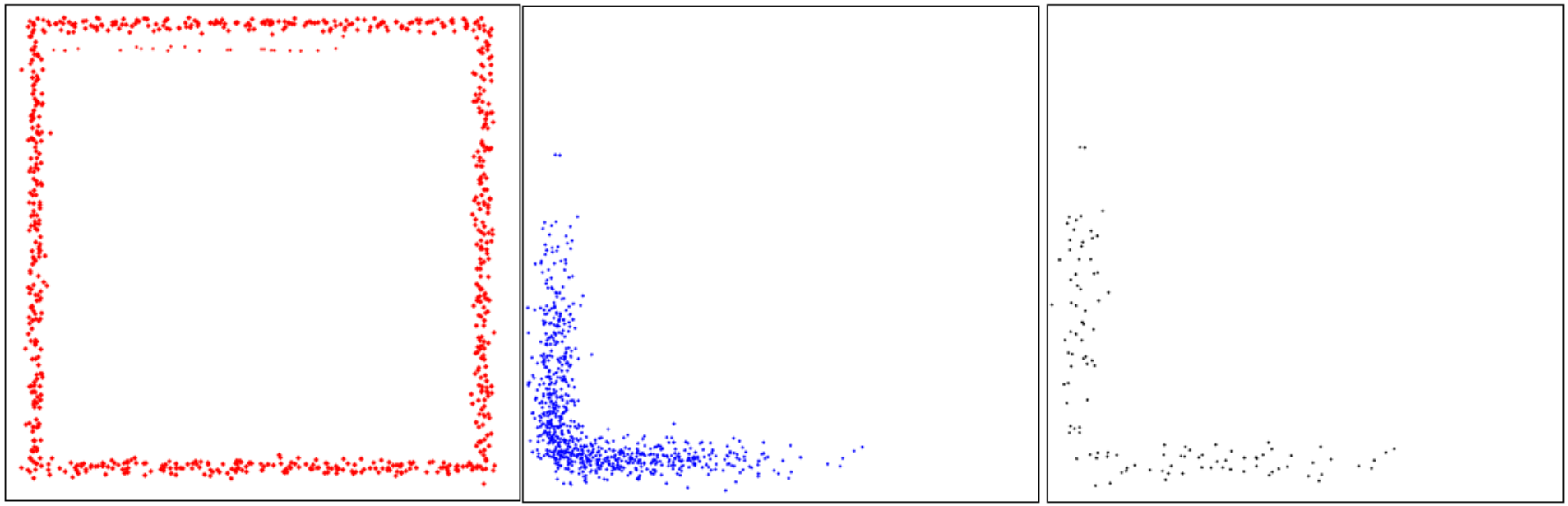
Resample by:

Iterate:

Choose a bin uniformly and at random

Choose a point from that bin uniformly and at random

Issue for LIDAR



Map

Vehicle observations

Stratified resample of observations

Biasing by normal can help

Resample the point cloud so that there are about the same number of points with each normal

How?

cut the unit sphere into bins,

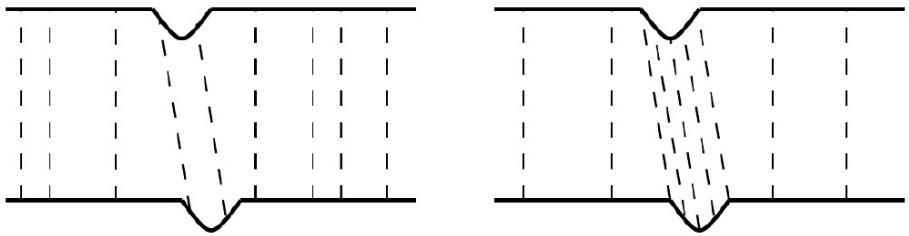
read points into bins

resample by:

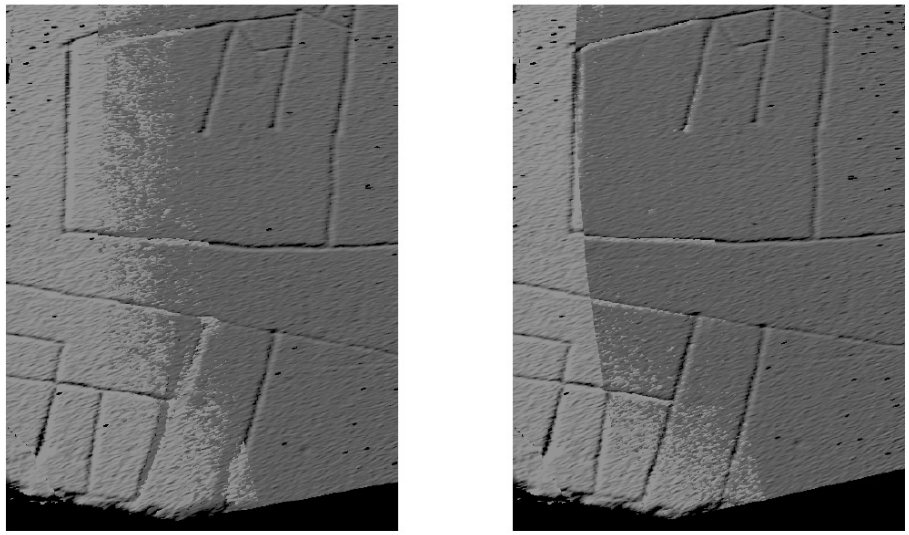
sample bin uniformly and at random

sample points in bin uniformly and at random

Biasing by normal can help



(a) (b)



(c) (d)