深蓝学院
shenlanxueyuan.com

3D Point Clouds
Lecture 9 – Registration

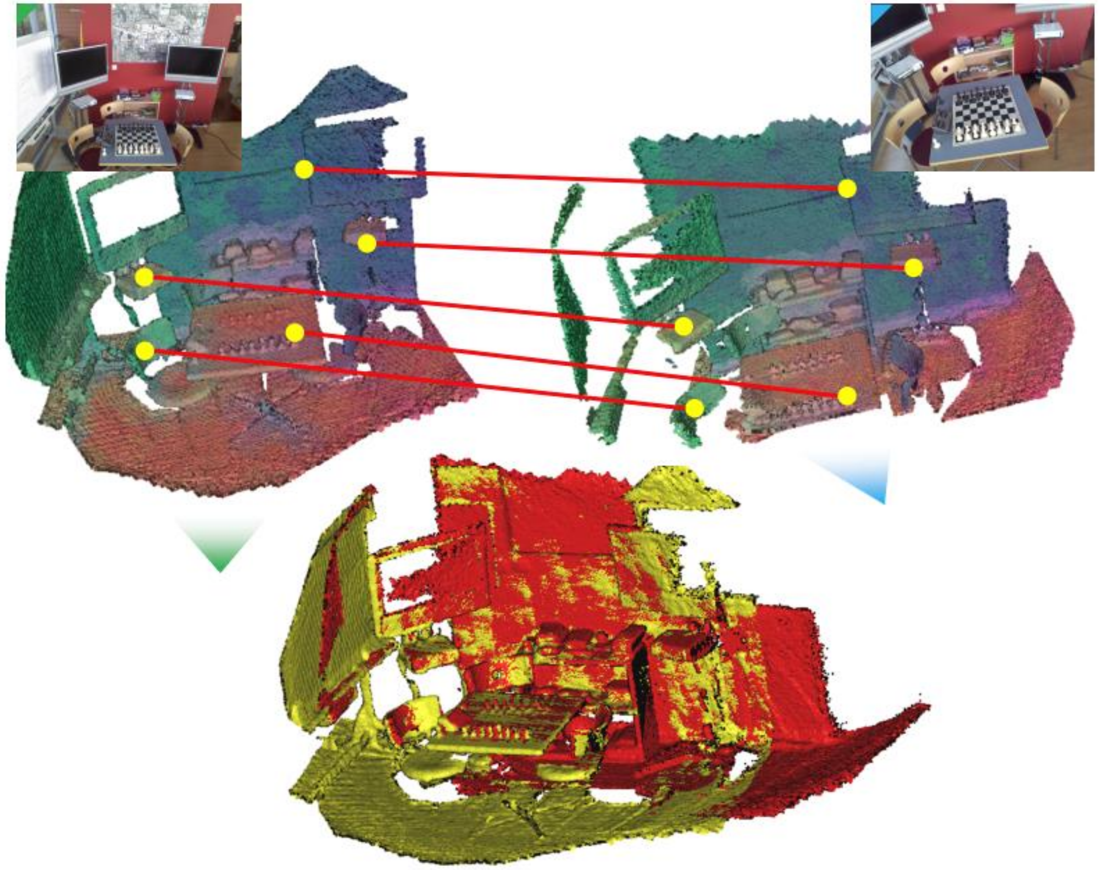
主讲人 黎嘉信

Aptiv 自动驾驶
新加坡国立大学 博士
清华大学 本科

1. Iterative Closest Point (ICP)

2. Normal Distribution Transform (NDT)

3. Feature Based Registration



Registration

- Find a transform to align two point clouds
- A transform consists of
  - Rotation R
  - Translation t

Source: PPFNet: Global Context Aware Local Features for Robust 3D Point Matching

Registration

- What is “transform”?
  - Translation
  - Rotation

- Applications:
  - Odometry / SLAM
  - Mapping
  - Loop Closure
  - Calibration
  - Object pose estimation
  - … …

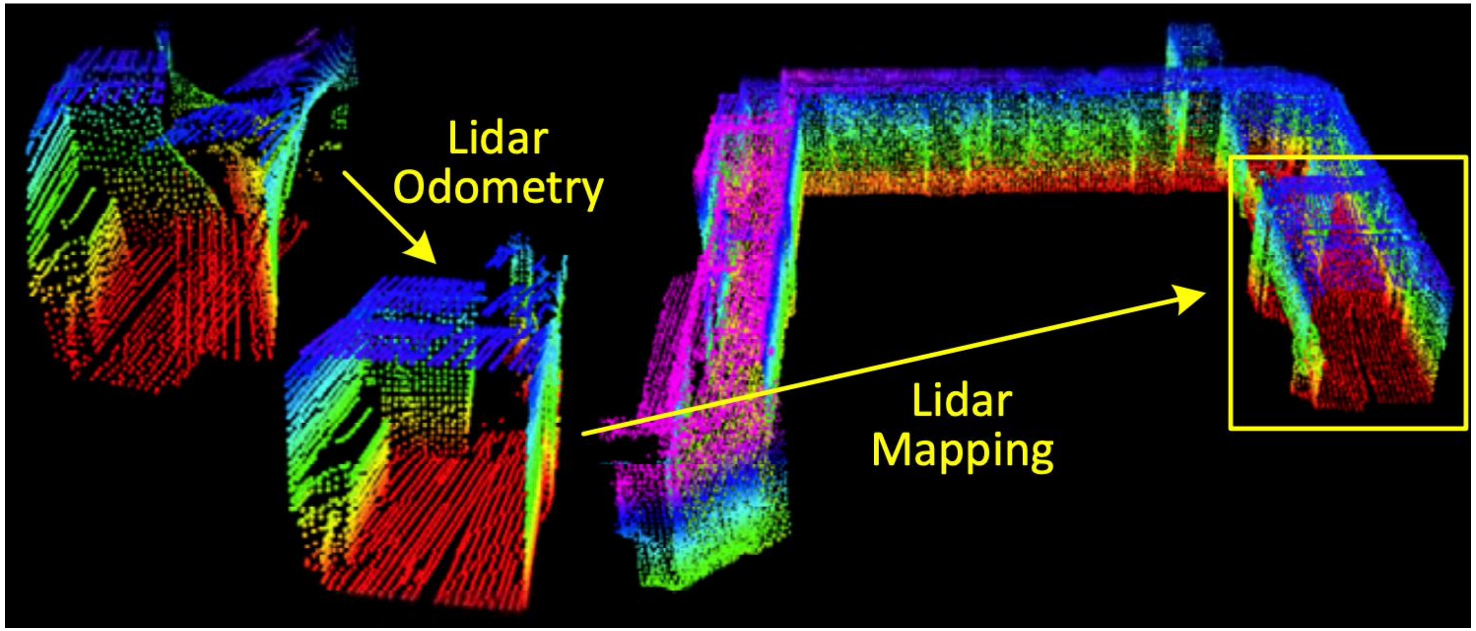


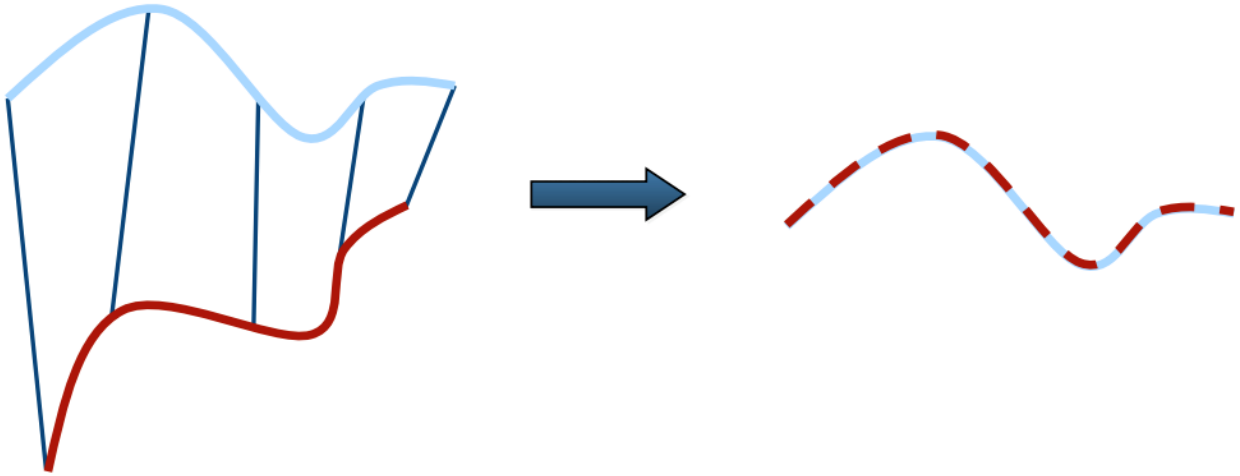
Image source: LOAM: Lidar Odometry and Mapping in Real-time, Zhang Ji, et.al.

Problem Definition

- Given two corresponding point sets:
  - $P = \{p_1, \cdots, p_{N_p}\}$, $p_i \in \mathbb{R}^3$, source
  - $Q = \{q_1, \cdots, q_{N_q}\}$, $q_i \in \mathbb{R}^3$, destination / target / reference
- Find the following that "best align" the point sets P, Q
  - rotation matrix $R \in \mathbb{R}^3, RR^T = I$
  - translation vector $t \in \mathbb{R}^3$
- What is "best align"?
  - If we have N correspondences between P, Q, we may minimize the mean squared error

$$E(R,t) = \frac{1}{N}\sum_{i=1}^{N}\|q_i - Rp_i - t\|^2$$

- ICP is iterating 2 steps
  1. Find correspondences between P, Q
  2. Minimize $E(R, t)$ to solve R, t
     - There is closed form solution given correspondences

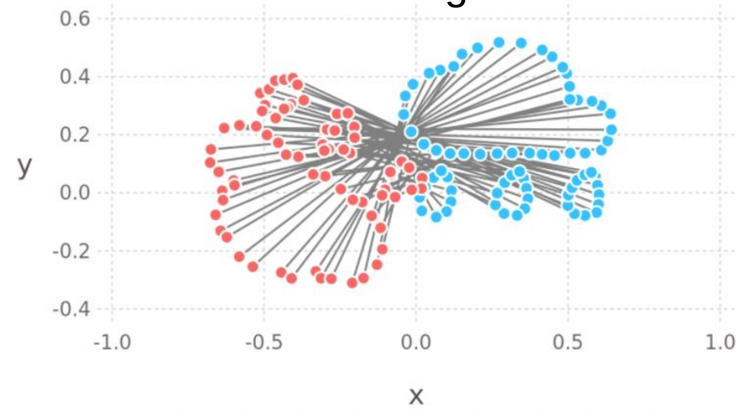


Iterative Closest Point (ICP)

- Given two corresponding point sets:
  - $P = \{p_1, \cdots, p_{N_p}\},\ p_i \in \mathbb{R}^3$, we are transforming P (source)
  - $Q = \{q_1, \cdots, q_{N_q}\},\ q_i \in \mathbb{R}^3$, assume Q is fixed (target)

1. Data association: N correspondences
   1. For each point p_i find the nearest neighbor in Q
   2. Remove outlier pairs, e.g., $\|p_i - q_i\|$ too large
2. $R, t = \arg_{R,t} \min E(R,t) = \arg_{R,t} \min \frac{1}{N} \sum_{i=1}^{N} \|q_i - Rp_i - t\|^2$
3. Check converge
   1. Evaluate convergence criteria
      1. $E(R,t)$ small enough
      2. $\Delta R, \Delta t$ small enough
   2. If not converged,
      1. $P \leftarrow RP + t$
      2. repeat Step 1-3

Example of perfect match, not nearest neighbor



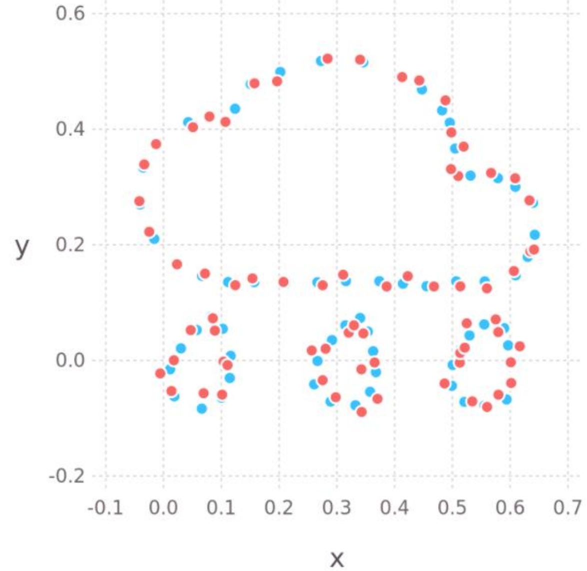


Image source: https://simonensemble.github.io/2018-10-27-orthogonal-procrustes/

- 普洛克路斯忒斯(Procrustes)是古希腊神话中的一个强盗。据公元前一世纪古希腊历史学家狄奥多(Diodoros,约公元前80-前29年)所编《历史丛书》记述：普洛克路斯忒斯开设黑店，拦截过路行人。他特意设置了两张铁床，一长一短，强迫旅客躺在铁床上，身矮者睡长床，强拉其躯体使与床齐；身高者睡短床，他用利斧把旅客伸出来的腿脚截短。由于他这种特殊的残暴方式，人称之为“铁床匪”。后来，希腊著名英雄忒修斯(Theseus)在前往雅典寻父途中，遇上了“铁床匪”，击败了这个拦路大盗。忒修斯以其人之道还治其人之身，强令身体魁梧的普洛克路斯忒斯躺在短床上，一刀砍掉了“铁床匪”伸出床外的下半肢，为民除了此害。

- Transform P to fit Q
  - $R, t = \arg_{R,t} \min E(R, t) = \arg_{R,t} \min \frac{1}{N} \sum_{i=1}^{N} \| q_i - R p_i - t \|^2$

- This problem can be extended to m dimension

$$A = [a_1, \cdots, a_N] \in \mathbb{R}^{m \times N},$$

$$B = [b_1, \cdots, b_N] \in \mathbb{R}^{m \times N},$$

$$\mathbf{1} = [1, \cdots, 1]^T \in \mathbb{R}^N$$

$$f(R, t) = \frac{1}{N} \sum_{i=1}^{N} \| b_i - R a_i - t \|^2 = \left\| B - \left(RA + t\mathbf{1}^{\boldsymbol{T}} \right) \right\|_F^2$$

$$\min_{R,t} f(R, t), s.t.\ RR^T = I_m$$

Frobenius Norm

$$f(R,t) = \frac{1}{N}\sum_{i=1}^{N}\|b_i - Ra_i - t\|^2 = \left\|B - \left(RA + t\mathbf{1}^{\boldsymbol{T}}\right)\right\|_F^2$$

- Matrix Frobenius Norm for matrix $A \in \mathbb{R}^{m\times n}$

$$\|A\|_{\mathrm{F}} = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2} = \sqrt{\mathrm{trace}(A^*A)} = \sqrt{\sum_{i=1}^{\min\{m,n\}} \sigma_i^2(A)}$$

Singular value of A

- Conjugate Transpose $A^* = A^T$
  - Only if A is real matrix
- Frobenius decomposition

$$\|A+B\|_{\mathrm{F}}^2 = \|A\|_{\mathrm{F}}^2 + \|B\|_{\mathrm{F}}^2 + 2\langle A,B\rangle_{\mathrm{F}}$$

- Where Frobenius inner product $\langle A,B\rangle_F$ is

$$\langle A,B\rangle_F = \sum_{ij} A_{ij}B_{ij} = \mathrm{tr}(A^TB) = \mathrm{tr}(AB^T)$$

$$f(R,t) = \frac{1}{N}\sum_{i=1}^{N}\|b_i - Ra_i - t\|^2 = \left\|B - \left(RA + t\mathbf{1}^{\boldsymbol{T}}\right)\right\|_F^2$$

- Trace of a square matrix
  - Sum of diagonal elements $\operatorname{tr}(\mathbf{A}) = \sum_{i=1}^{n} a_{ii} = a_{11} + a_{22} + \cdots + a_{nn}$
- Trace is a linear mapping

$$\operatorname{tr}(\mathbf{A}+\mathbf{B}) = \operatorname{tr}(\mathbf{A}) + \operatorname{tr}(\mathbf{B})$$

$$\operatorname{tr}(c\mathbf{A}) = c\operatorname{tr}(\mathbf{A})$$

$$\operatorname{tr}(\mathbf{A}) = \operatorname{tr}(\mathbf{A}^\top)$$

- Trace of a product

$$\operatorname{tr}(\mathbf{A}^\top\mathbf{B}) = \operatorname{tr}(\mathbf{A}\mathbf{B}^\top) = \operatorname{tr}(\mathbf{B}^\top\mathbf{A}) = \operatorname{tr}(\mathbf{B}\mathbf{A}^\top) = \sum_{i,j} A_{ij}B_{ij}$$

$$\operatorname{tr}(\mathbf{b}\mathbf{a}^T) = \mathbf{a}^T\mathbf{b}$$

- Cyclic property

$$\operatorname{tr}(\mathbf{ABCD}) = \operatorname{tr}(\mathbf{BCDA}) = \operatorname{tr}(\mathbf{CDAB}) = \operatorname{tr}(\mathbf{DABC})$$

- Transform P to fit Q
  - $R, t = \arg_{R,t} \min E(R, t)$

- This problem can be extended to m dimension

  Transform A to fit B

  $A = [a_1, \cdots, a_N] \in \mathbb{R}^{m \times N}$,

  $B = [b_1, \cdots, b_N] \in \mathbb{R}^{m \times N}$,

  $\mathbf{1} = [1, \cdots, 1]^T \in \mathbb{R}^N$

$$f(R, t) = \frac{1}{N} \sum_{i=1}^{N} \|b_i - Ra_i - t\|^2 = \left\| B - \left(RA + t\mathbf{1}^{\boldsymbol{T}}\right) \right\|_F^2$$

$$\min_{R,t} f(R, t), s.t.\ RR^T = I_m$$

Solution:

Normalize A, B into A', B' by subtracting the mean

$$L = I_N - \frac{1}{N}\mathbf{1}\mathbf{1}^T$$

$$A' = AL$$

$$B' = BL$$

Perform SVD for $B'A'^T$,

$$B'A'^T = U\Sigma V^T$$

The optimization solution is,

$$R^* = UV^T$$

$$t^* = \frac{1}{N}(B - R^*A)\mathbf{1}$$

$\frac{A}{N}\mathbf{1}, \frac{B}{N}\mathbf{1}$ are mean of $\{a_i\}, \{b_i\}$ respectively

Procrustes Transformation

$$\min_{R,t} f(R,t)\,, s.t.\ RR^T = I_m$$

$$\begin{aligned} f(R,t) &= \|B - RA - t\mathbf{1}^T\|_F^2 \\ &= \|(B - RA) + (-t\mathbf{1}^T)\|_F^2 \\ &= \|B - RA\|_F^2 + \| - t\mathbf{1}^T\|_F^2 + 2\langle B - RA, -t\mathbf{1}^T\rangle \\ &= \|B - RA\|_F^2 + Nt^Tt - 2\mathrm{tr}\big((B - RA)\mathbf{1}t^T\big) \end{aligned}$$

$$\boxed{\|A\|_{\mathrm{F}} = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}}$$

$$\boxed{\|A + B\|_{\mathrm{F}}^2 = \|A\|_{\mathrm{F}}^2 + \|B\|_{\mathrm{F}}^2 + 2\langle A, B\rangle_{\mathrm{F}}}$$

$$\boxed{\langle A, B\rangle_F = \mathrm{tr}(AB^T)}$$

- Parameters to be optimized:
  - Rotation $R \in \mathbb{R}^{m\times m}, RR^T = I_m$
  - Translation $t \in \mathbb{R}^m$

$$\min_{R,t} f(R,t)\,, s.t.\ RR^T = I_m$$

- Cost function

$$f(R,t) = \|B - RA\|_F^2 + Nt^T t - 2\mathrm{tr}\big((B - RA)\mathbf{1}t^T\big)$$

- Look at t first
  - $f(R,t)$ is quadratic on t
  - Minimum achieved at zero first-order derivative

$$\frac{\partial f}{\partial t} = 0$$

$$2Nt - 2(B - RA)\mathbf{1} = 0$$

$$t = \frac{1}{N}(B - RA)\mathbf{1}$$

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{X}}\mathrm{Tr}(\mathbf{X}) &= \mathbf{I} \\
\frac{\partial}{\partial \mathbf{X}}\mathrm{Tr}(\mathbf{X}\mathbf{A}) &= \mathbf{A}^T \\
\frac{\partial}{\partial \mathbf{X}}\mathrm{Tr}(\mathbf{A}\mathbf{X}\mathbf{B}) &= \mathbf{A}^T\mathbf{B}^T \\
\frac{\partial}{\partial \mathbf{X}}\mathrm{Tr}(\mathbf{A}\mathbf{X}^T\mathbf{B}) &= \mathbf{B}\mathbf{A} \\
\frac{\partial}{\partial \mathbf{X}}\mathrm{Tr}(\mathbf{X}^T\mathbf{A}) &= \mathbf{A} \\
\boxed{\frac{\partial}{\partial \mathbf{X}}\mathrm{Tr}(\mathbf{A}\mathbf{X}^T) = \mathbf{A}} \\
\frac{\partial}{\partial \mathbf{X}}\mathrm{Tr}(\mathbf{A}\otimes\mathbf{X}) &= \mathrm{Tr}(\mathbf{A})\mathbf{I}
\end{aligned}$$

Source: The Matrix Cookbook, Kaare Brandt Petersen, Michael Syskind Pedersen

$$\min_{R,t} f(R,t),\ s.t.\ RR^T = I_m$$

- Cost Function $f(R,t) = \|B - RA\|_F^2 + Nt^Tt - 2\text{tr}\left((B - RA)\mathbf{1}t^T\right)$
- Optimal $t = \frac{1}{N}(B - RA)\mathbf{1}$
- Substitute t into $f(R,t)$,

$$\begin{aligned} f(R,t) &= \|B - RA\|_F^2 + Nt^Tt - 2\text{tr}\left((B - RA)\mathbf{1}t^T\right) \\ &= \|B - RA\|_F^2 + \boxed{\frac{1}{N}\mathbf{1}^T(B - RA)^T(B - RA)\mathbf{1}} - \boxed{2\text{tr}\left((B - RA)\mathbf{1}\frac{1}{N}\mathbf{1}^T(B - RA)^T\right)} \end{aligned}$$

- Consider the green and blue part separately

Procrustes Transformation – Green part

$$
\begin{aligned}
&\frac{1}{N}\mathbf{1}^T(B-RA)^T(B-RA)\mathbf{1} \\
&= \frac{1}{N}\begin{bmatrix}1 & \cdots & 1\end{bmatrix}\begin{bmatrix}(b_1-Ra_1)^T \\ \vdots \\ (b_N-Ra_N)^T\end{bmatrix}\begin{bmatrix}(b_1-Ra_1) & \cdots & (b_N-Ra_N)\end{bmatrix}\begin{bmatrix}1 \\ \vdots \\ 1\end{bmatrix} \\
&= \frac{1}{N}\Big(\big((b_1-Ra_1)^T+\cdots+(b_N-Ra_N)^T\big)\big((b_1-Ra_1)+\cdots+(b_N-Ra_N)\big)\Big) \\
&= \frac{1}{N}\left(\sum_{i=1}^{N}(b_i-Ra_i)^T\right)\left(\sum_{i=1}^{N}(b_i-Ra_i)\right) \\
&= \frac{1}{N}N(\mu_b-R\mu_a)^T N(\mu_b-R\mu_a) \\
&= N\|\mu_b-R\mu_a\|^2
\end{aligned}
$$

Procrustes Transformation – Blue part

$$
\begin{aligned}
&2\mathrm{tr}\left((B - RA)\mathbf{1}\frac{1}{N}\mathbf{1}^T(B - RA)^T\right) \\
&= 2\frac{1}{N}\mathrm{tr}\left(\begin{bmatrix}(b_1 - Ra_1) & \cdots & (b_N - Ra_N)\end{bmatrix}\begin{bmatrix}1 \\ \vdots \\ 1\end{bmatrix}\begin{bmatrix}1 & \cdots & 1\end{bmatrix}\begin{bmatrix}(b_1 - Ra_1)^T \\ \vdots \\ (b_N - Ra_N)^T\end{bmatrix}\right) \\
&= 2\frac{1}{N}\mathrm{tr}\left(\left((b_1 - Ra_1) + \cdots + (b_N - Ra_N)\right)\left((b_1 - Ra_1)^T + \cdots + (b_N - Ra_N)^T\right)\right) \\
&= 2\frac{1}{N}\mathrm{tr}\left(\left(\sum_{i=1}^{N}(b_i - Ra_i)\right)\left(\sum_{i=1}^{N}(b_i - Ra_i)^T\right)\right) \\
&= 2\frac{1}{N}\mathrm{tr}\left(N(\mu_b - R\mu_a)N(\mu_b - R\mu_a)^T\right) \\
&= 2N\mathrm{tr}\left((\mu_b - R\mu_a)(\mu_b - R\mu_a)^T\right) \\
&= 2N\|\mu_b - R\mu_a\|^2
\end{aligned}
$$

$$\min_{R,t} f(R,t)\,, s.t.\ RR^T = I_m$$

- Cost Function

$$
\begin{aligned}
f(R,t) &= \|B-RA\|_F^2 + Nt^Tt - 2\mathrm{tr}\left((B-RA)\mathbf{1}t^T\right) \\
&= \|B-RA\|_F^2 + \boxed{\frac{1}{N}\mathbf{1}^T(B-RA)^T(B-RA)\mathbf{1}} - \boxed{2\mathrm{tr}\left((B-RA)\mathbf{1}\frac{1}{N}\mathbf{1}^T(B-RA)^T\right)} \\
&= \sum_{i=1}^N \|b_i - Ra_i\|^2 + \sum_{i=1}^N \|\mu_b - R\mu_a\|^2 - 2\sum_{i=1}^N (b_i - Ra_i)^T(\mu_b - R\mu_a) \\
&= \sum_{i=1}^N \|(b_i - Ra_i) - (\mu_b - R\mu_a)\|^2 \\
&= \sum_{i=1}^N \|(b_i - \mu_b) - R(a_i - \mu_a)\|^2 \\
&= \sum_{i=1}^N \|b_i' - Ra_i'\|^2 \\
&= \|B' - RA'\|_F^2
\end{aligned}
$$

$$\min_{R,t} f(R,t)\,,s.t.\ RR^T = I_m$$

- Cost function $f(R,t) = \|B' - RA'\|_F^2$

$$\begin{aligned}
f(R,t) &= \|B' - RA'\|_F^2 \\
&= \|B'\|_F^2 + \|RA'\|_F^2 - 2\langle B', RA'\rangle \\
&= \|B'\|_F^2 + \|A'\|_F^2 - 2\mathrm{tr}(RA'B'^T) \\
&= \|B'\|_F^2 + \|A'\|_F^2 - 2\mathrm{tr}(RV\Sigma U^T) \\
&= \|B'\|_F^2 + \|A'\|_F^2 - 2\mathrm{tr}(U^T RV\Sigma) \\
&= \|B'\|_F^2 + \|A'\|_F^2 - 2\mathrm{tr}(T\Sigma) \\
&= \|B'\|_F^2 + \|A'\|_F^2 - 2\sum_{i=1}^{m} T_{ii}\sigma_i
\end{aligned}$$

- Original problem: $\min_{R,t} f(R,t)\,, s.t.\ RR^T = I_m$

$$f(R,t) = \|B'\|_F^2 + \|A'\|_F^2 - 2\sum_{i=1}^{m} T_{ii}\sigma_i$$

- Now it equals to: $\max_{R,t}\ (\sum_{i=1}^{m} T_{ii}\sigma_i)\,, s.t.\ TT^T = I_m$
- U, R, V are orthogonal → $T = U^T RV$ is orthogonal → $|T_{ii}| \leq 1$
- The maximum is achieved when $T_{ii} = 1$, i.e., $T = I_m$

$$T = U^T RV = I_m$$

- Note that Finally,

$$R = UV^T$$

$$t = \frac{1}{N}(B - RA)\mathbf{1}$$

Procrustes Transformation Problem

- Transform P to fit Q
  - $R, t = \arg_{R,t} \min E(R, t)$

- This problem can be extended to m dimension

  Transform A to fit B

$$A = [a_1, \cdots, a_N] \in \mathbb{R}^{m \times N},$$

$$B = [b_1, \cdots, b_N] \in \mathbb{R}^{m \times N},$$

$$\mathbf{1} = [1, \cdots, 1]^T \in \mathbb{R}^N$$

$$f(R, t) = \frac{1}{N} \sum_{i=1}^{N} \|b_i - Ra_i - t\|^2 = \left\| B - \left(RA + t\mathbf{1}^T\right) \right\|_F^2$$

$$\min_{R,t} f(R, t), s.t.\ RR^T = I_m$$

Solution:

Normalize A, B into A', B' by subtracting the mean

$$L = I_N - \frac{1}{N}\mathbf{1}\mathbf{1}^T$$

$$A' = AL$$

$$B' = BL$$

Perform SVD for $B'A'^T$,

$$B'A'^T = U\Sigma V^T$$

The optimization solution is,

$$R^* = UV^T$$

$$t^* = \frac{1}{N}(B - R^*A)\mathbf{1}$$

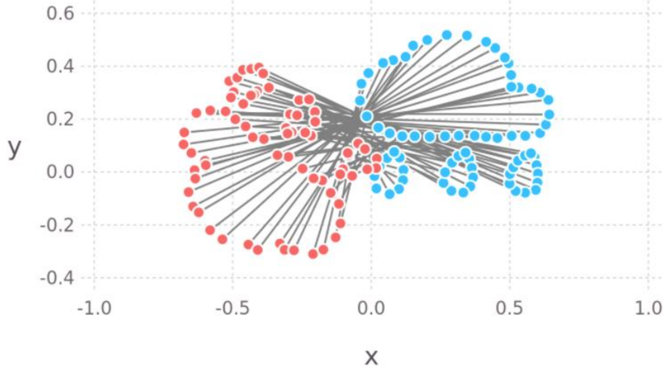
$\frac{A}{N}\mathbf{1}, \frac{B}{N}\mathbf{1}$ are mean of $\{a_i\}, \{b_i\}$ respectively

Iterative Closest Point (ICP)

- Given two corresponding point sets:
  - $P = \{p_1, \cdots, p_{N_p}\}$, $p_i \in \mathbb{R}^3$, we are transforming P (source)
  - $Q = \{q_1, \cdots, q_{N_q}\}$, $q_i \in \mathbb{R}^3$, assume Q is fixed (target)

1. Data association: N correspondences
   1. For each point p_i find the nearest neighbor in Q
   2. Remove outlier pairs, e.g., $\|p_i - q_i\|$ too large
2. $R, t = \arg_{R,t} \min E(R, t) = \arg_{R,t} \min \frac{1}{N} \sum_{i=1}^{N} \|q_i - Rp_i - t\|^2$
   1. Compute center $\mu_p = \frac{1}{N} \sum_{i=1}^{N} p_i$, $\mu_q = \frac{1}{N} \sum_{i=1}^{N} q_i$
   2. $P' = \{p_i - \mu_p\}, Q' = \{q_i - \mu_q\}$
   3. $Q' P'^T = U \Sigma V^T$
   4. $R = UV^T, t = \mu_q - R\mu_p$
3. Check converge.
   1. Evaluate convergence criteria
      1. $E(R, t)$ small enough
      2. $\Delta R, \Delta t$ small enough
   2. If not converged,
      1. $P \leftarrow RP + t$
      2. repeat Step 1-3

Example of perfect match, not nearest neighbor



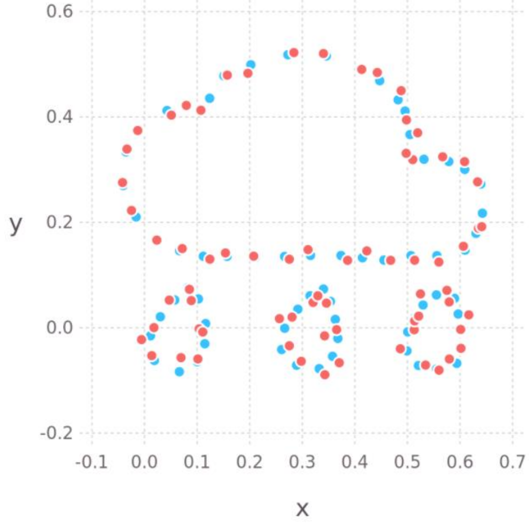


Image source: https://simonensemble.github.io/2018-10-27-orthogonal-procrustes/

Iterative Closest Point (ICP)

- Given two corresponding point sets:
  - $P = \{p_1, \cdots, p_{N_p}\}$, $p_i \in \mathbb{R}^3$, we are transforming P (source)
  - $Q = \{q_1, \cdots, q_{N_q}\}$, $q_i \in \mathbb{R}^3$, assume Q is fixed (target)

1. Data association: N correspondences
   1. For each point p_i find the nearest neighbor in Q
   2. Remove outlier pairs, e.g., $\|p_i - q_i\|$ too large
2. $R, t = \arg_{R,t} \min E(R, t) = \arg_{R,t} \min \frac{1}{N} \sum_{i=1}^{N} \|q_i - Rp_i - t\|^2$
   1. Compute center $\mu_p = \frac{1}{N} \sum_{i=1}^{N} p_i$, $\mu_q = \frac{1}{N} \sum_{i=1}^{N} q_i$
   2. $P' = \{p_i - \mu_p\}, Q' = \{q_i - \mu_q\}$
   3. $Q' P'^T = U \Sigma V^T$
   4. $R = UV^T, t = \mu_q - R\mu_p$
3. Check converge.
   1. Evaluate convergence criteria
      1. $E(R, t)$ small enough
      2. $\Delta R, \Delta t$ small enough
   2. If not converged,
      1. $P \leftarrow RP + t$
      2. repeat Step 1-3

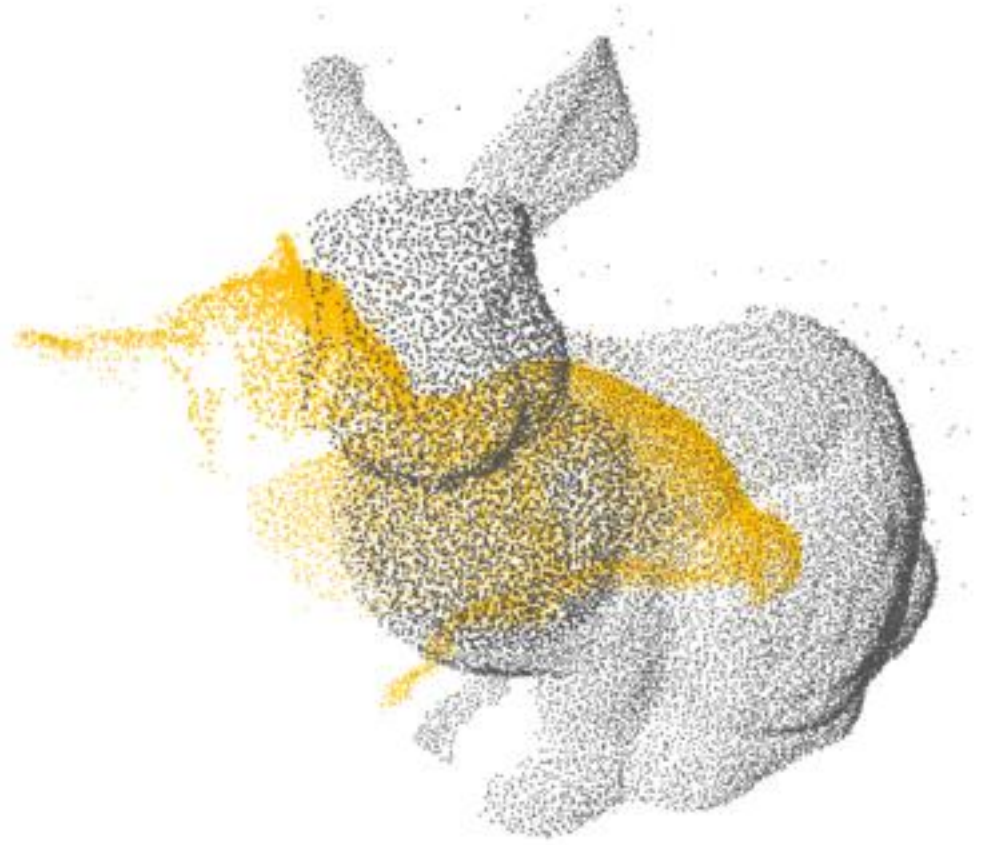
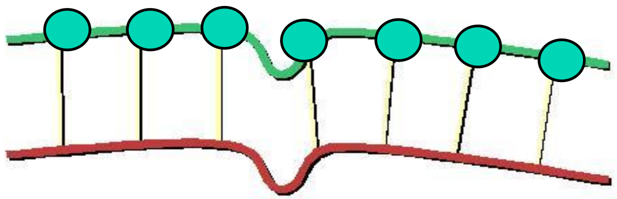


Image source: https://www.cs.cmu.edu/~halismai/cicp/

ICP Variants

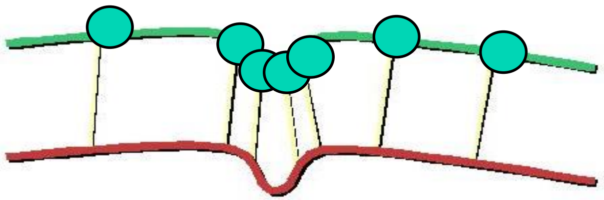
1. Point Subsets (from one or both point sets)
    1. Random Sample
    2. Voxel Grid Sample
    3. Normal Space Sampling (NSS)
    4. Feature detection
2. Data association
    1. Nearest neighbor – kd-tree/octree for acceleration
    2. Normal shooting
    3. Projection
    4. Feature descriptor matching (compatible point)
3. Outlier Rejection
    1. Remove correspondence with high distance
    2. Remove worst x% of correspondences
4. Loss function
    1. Point-to-Point
    2. Point-to-Plane

- Uniform sampling
  - Some feature areas are ignored
  - E.g., the red/green curve can slide left or right without increasing the cost



Uniform Sampling

- Normal Space Sampling (NSS)
  - Pay more attention to minority surface normals
  - E.g., more points are sampled at the protruding area



Normal Space Sampling

- Nearest neighbor search

$$q_i = \arg\min_{q_i \in Q} \|p_i - q_i\|^2$$

  - Generally it works well

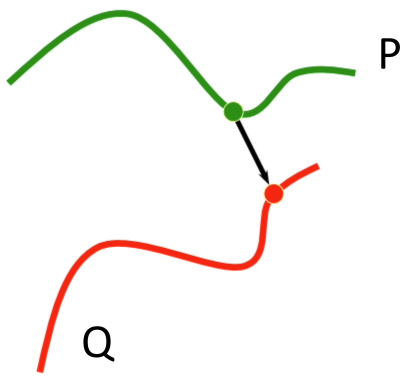
- Normal Shooting
  - Find a point that is closest to the surface normal vector

$$q_i' = q_i - p_i$$

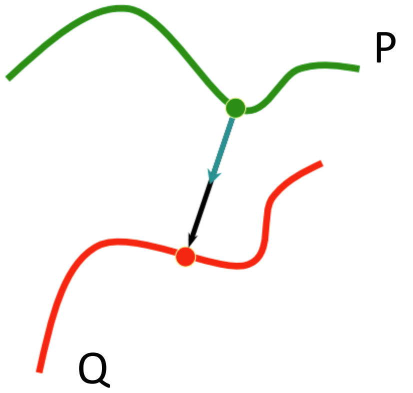
$$q_i = \arg\min_{q_i \in Q} \left\| q_i' - \left({q_i'}^T n_{p_i} \right) n_{p_i} \right\|^2$$

  - Works for smooth structures
  - Doesn't work for points with un-reliable surface normals
    - E.g., complex structures

Nearest Neighbor

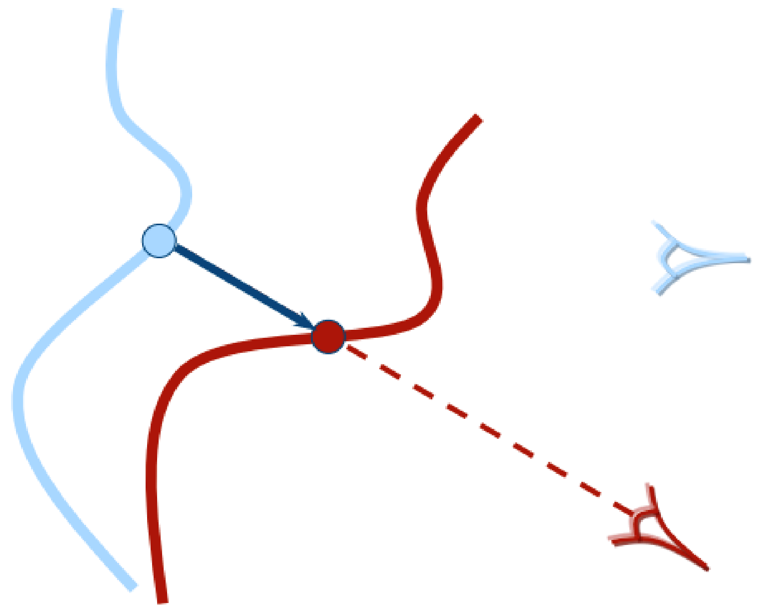


Normal Shooting

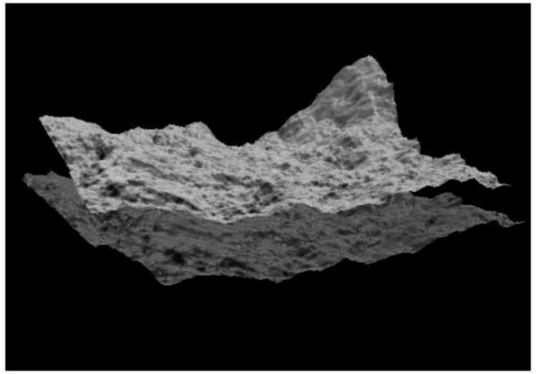


- Works for depth image (RGBD)

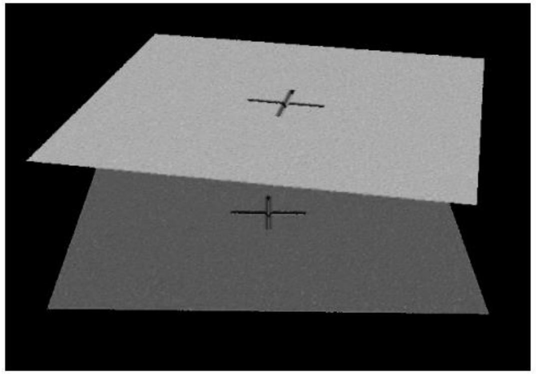
- Approximate the NN search by
  1. Project the blue point (p_i) into the red frame to get the pixel location
  2. Get the red point (q_i) by getting the depth of that pixel location.



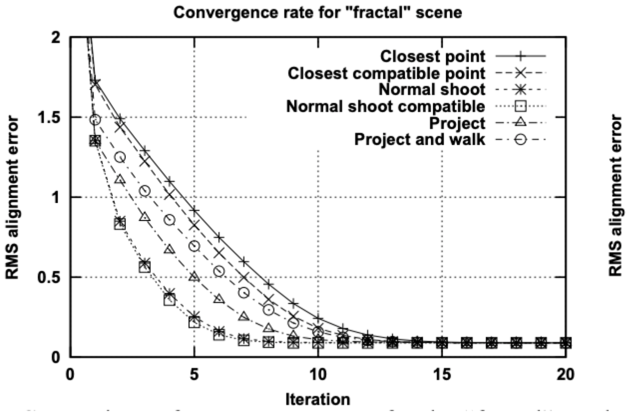
Various Data Association

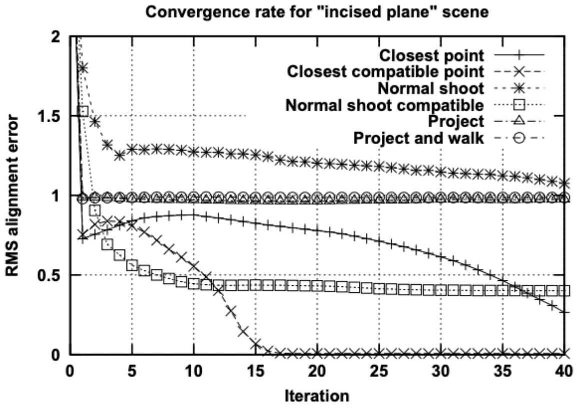


(b) Fractal landscape



(c) Incised plane



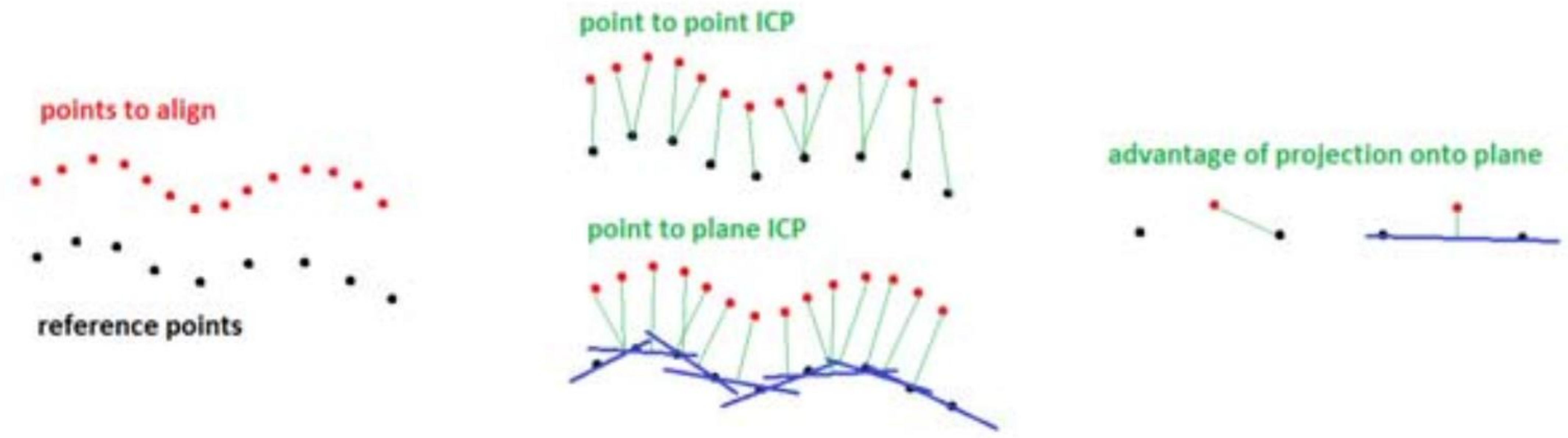


- Closest point is robust in general.
- Other associations may be faster, but not as stable.

Source: Efficient Variants of the ICP Algorithm, Szymon Rusinkiewicz, Marc Levoy

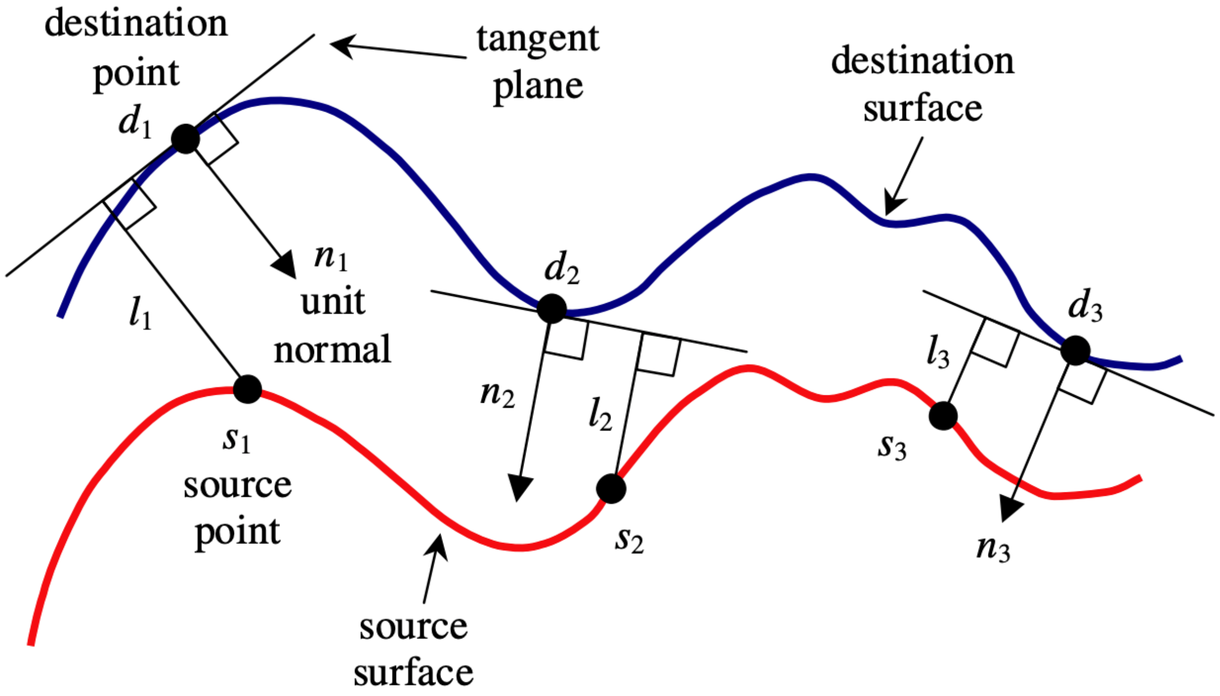
Loss Function - Point-to-Plane

- Point-to-Plane cost function allows flat regions to slide along each other



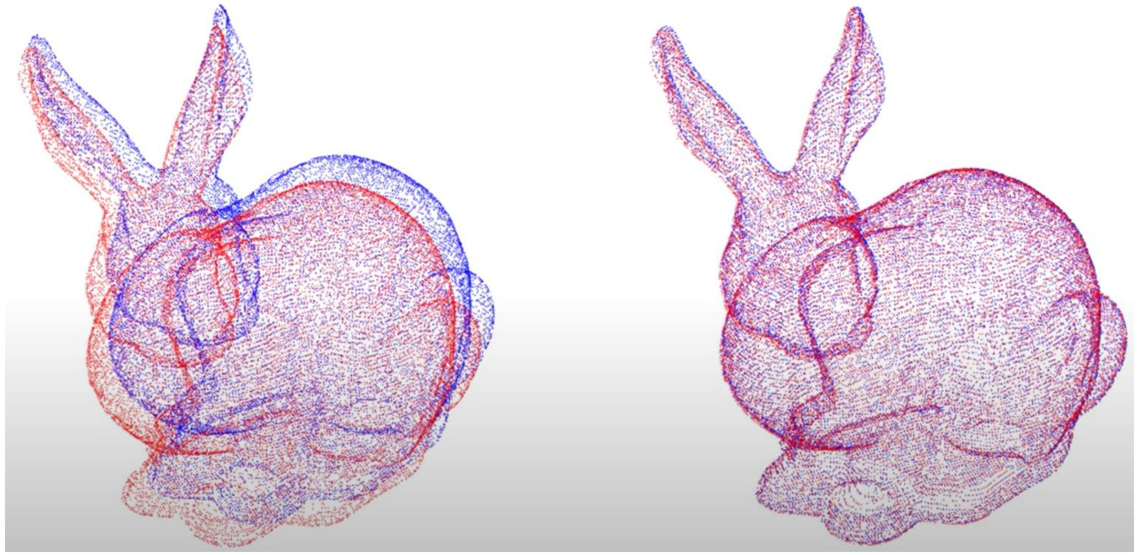
Point-to-Plane: Object modelling by registration of multiple range images, Y. Chen and G. G. Medioni, 1992
Image source: GPGPU computation in mobile robot applications

- Point-to-Point
  - Distance between two points
  - $E(R,t) = \frac{1}{N}\sum_{i=1}^{N}\|q_i - Rp_i - t\|^2$
- Point-to-Plane
  - Distance between source point p_i and the local surface of destination point q_i
  - n_i is the surface normal at destination point q_i
  - $E(R,t) = \sum_{i=1}^{N}\left((Rp_i + t - q_i)^T n_i\right)^2$



- Point-to-Plane cost function allows flat regions slide to along each other
- Point-to-Plane usually converges faster than Point-to-Point
  - Takes less iterations
- Point-to-Plane is slower in each iteration, and requires surface normal

Point-to-Point
Iteration 4



Point-to-Plane
Iteration 4

Image source: https://www.youtube.com/watch?v=LcghboLgTiA

- Point-to-Plane
  - n_i is the surface normal at point q_i
  - $E(R, t) = \sum_{i=1}^{N}\left((Rp_i + t - q_i)^T n_i\right)^2$
  - There is NO analytical solution.
  - How? Least Square optimization!
- How to representation rotation matrix?
  - 9 elements - over-parameterization. Subjected to constraint $RR^T = I$
    - Constrained optimization is more troublesome than unconstrained ones.
  - Euler angles
  - Angle-axis
  - Quaternion
  - Exponential map / lie-algebra

- Represent $R \in \mathbb{R}^3$ by angles
  - x axis by α, y axis by β, z axis by γ

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$$

$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}$$

$$R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R = R_z(\gamma)R_y(\beta)R_x(\alpha) = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

$$\begin{aligned}
r_{11} &= \cos\gamma\cos\beta, \\
r_{12} &= -\sin\gamma\cos\alpha + \cos\gamma\sin\beta\sin\alpha, \\
r_{13} &= \sin\gamma\sin\alpha + \cos\gamma\sin\beta\cos\alpha, \\
r_{21} &= \sin\gamma\cos\beta, \\
r_{22} &= \cos\gamma\cos\alpha + \sin\gamma\sin\beta\sin\alpha, \\
r_{23} &= -\cos\gamma\sin\alpha + \sin\gamma\sin\beta\cos\alpha, \\
r_{31} &= -\sin\beta, \\
r_{32} &= \cos\beta\sin\alpha, \\
r_{33} &= \cos\beta\cos\alpha.
\end{aligned}$$

Source: Linear Least-Squares Optimization for Point-to-Plane ICP Surface Registration. Kok-Lim Low, NUS

- R is too complicated for optimization
  - Simplify by approximation
  - In each iteration we may assume the transformation is small
    - $\alpha, \beta, \gamma \to 0$
    - $\cos\theta \approx 1, \sin\theta \approx \theta, \theta^2 \approx 0$, if $\theta \approx 0$

$$R \approx \begin{bmatrix} 1 & \alpha\beta - \gamma & \alpha\gamma + \beta \\ \gamma & \alpha\beta\gamma + 1 & \beta\gamma - \alpha \\ -\beta & \alpha & 1 \end{bmatrix} \approx \begin{bmatrix} 1 & -\gamma & \beta \\ \gamma & 1 & -\alpha \\ -\beta & \alpha & 1 \end{bmatrix}$$

- Point-to-Plane cost function is linear about the unknowns
  - $\alpha, \beta, \gamma, t_x, t_y, t_z$

$$E(R,t) = \sum_{i=0}^{N} \left((Rp_i + t - q_i)^T n_i\right)^2$$
$$= \sum_{i=0}^{N} \left(\left(\begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_i \\ 1 \end{bmatrix} - \begin{bmatrix} q_i \\ 1 \end{bmatrix} \right)^T \begin{bmatrix} n_i \\ 0 \end{bmatrix} \right)^2$$

$$R \approx \begin{bmatrix} 1 & -\gamma & \beta \\ \gamma & 1 & -\alpha \\ -\beta & \alpha & 1 \end{bmatrix}$$

- Re-write it into the form of $Ax = b$
  - $A \in \mathbb{R}^{N\times 6},\ b \in \mathbb{R}^N$
  - $x = [\alpha, \beta, \gamma, t_x, t_y, t_z]^T$
  - $\hat{x} = (A^T A)^{-1} A^T b$, assume A is full column rank

Loss Function - Point-to-Plane

- $E(R,t) = \sum_{i=1}^{N}\left((Rp_i + t - q_i)^T n_i\right)^2 = \|Ax - b\|^2$
  - Look at i-th element

$$
\begin{aligned}
(Rp_i + t - q_i)^T n_i = {} & (n_{iz}p_{iy} - n_{iy}p_{iz})\alpha + (n_{ix}p_{iz} - n_{iz}p_{ix})\beta + (n_{iy}p_{ix} - n_{ix}p_{iy})\gamma \\
& + n_{ix}t_x + n_{iy}t_y + n_{iz}t_z \\
& - (n_{ix}q_{ix} + n_{iy}q_{iy} + n_{iz}q_{iz} - n_{ix}p_{ix} - n_{iy}p_{iy} - n_{iz}p_{iz})
\end{aligned}
$$

$$
x = \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ t_x \\ t_y \\ t_z \end{bmatrix}
$$

$$
A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & n_{1x} & n_{1y} & n_{1z} \\ a_{21} & a_{22} & a_{23} & n_{2x} & n_{2y} & n_{2z} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{N1} & a_{N2} & a_{N3} & n_{Nx} & n_{Ny} & n_{Nz} \end{bmatrix}
$$

$$
\begin{aligned}
a_{i1} &= n_{iz}p_{iy} - n_{iy}p_{iz} \\
a_{i2} &= n_{ix}p_{iz} - n_{iz}p_{ix} \\
a_{i3} &= n_{iy}p_{iz} - n_{ix}p_{iy}
\end{aligned}
$$

$$
b = \begin{bmatrix} n_{1x}q_{1x} + n_{1y}q_{1y} + n_{1z}q_{1z} - n_{1x}p_{1x} - n_{1y}p_{1y} - n_{1z}p_{1z} \\ n_{2x}q_{2x} + n_{2y}q_{2y} + n_{2z}q_{2z} - n_{2x}p_{2x} - n_{2y}p_{2y} - n_{2z}p_{2z} \\ \vdots \\ n_{Nx}q_{Nx} + n_{Ny}q_{Ny} + n_{Nz}q_{Nz} - n_{Nx}p_{Nx} - n_{Ny}p_{Ny} - n_{Nz}p_{Nz} \end{bmatrix}
$$

ICP – Point-to-Plane

- Given two corresponding point sets:
  - $P = \left\{p_1, \cdots, p_{N_p}\right\}$, $p_i \in \mathbb{R}^3$, we are transforming P (source)
  - $Q = \left\{q_1, \cdots, q_{N_q}\right\}$, $q_i \in \mathbb{R}^3$, assume Q is fixed (target)

1. Data association: N correspondences
   1. For each point p_i find the nearest neighbor in Q
   2. Remove outlier pairs, e.g., $\|p_i - q_i\|$ too large
2. $R, t = \arg_{R,t} \min E(R, t) = \arg_{R,t} \min \sum_{i=1}^{N} \left((Rp_i + t - q_i)^T n_i\right)^2$
   1. $\hat{x} = \arg\min_{x} E(x) = \|Ax - b\|^2 = (A^T A)^{-1} A^T b$
   2. $\hat{x} = \left[\alpha, \beta, \gamma, t_x, t_y, t_z\right]^T$
   3. Compute R, t from $\hat{x}$
3. Check converge.
   1. Evaluate convergence criteria
      1. $E(R, t)$ small enough
      2. $\Delta R, \Delta t$ small enough
   2. If not converged,
      1. $P \leftarrow RP + t$
      2. repeat Step 1-3

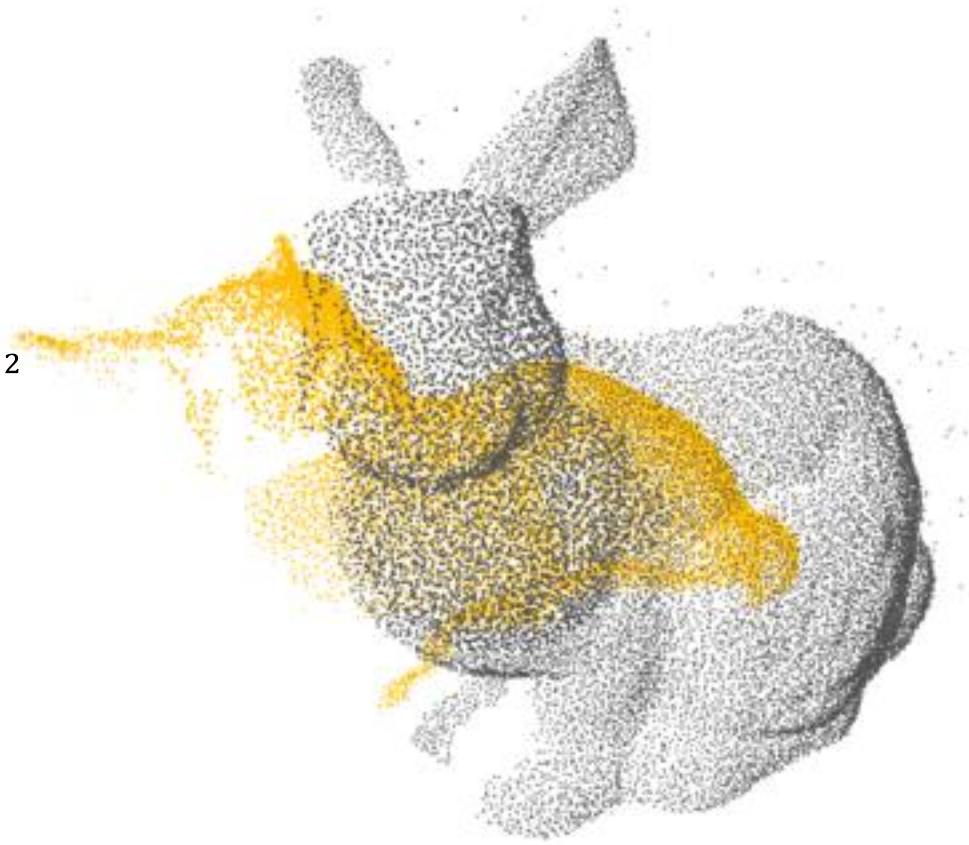
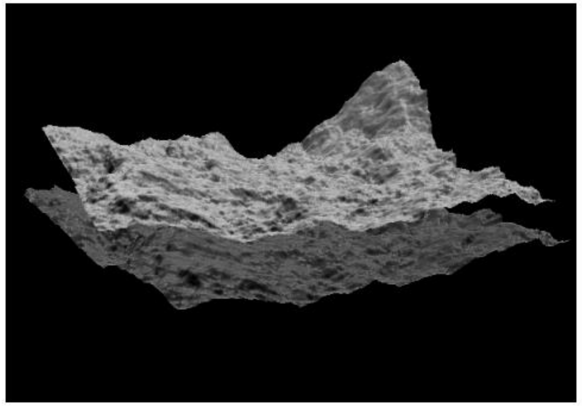
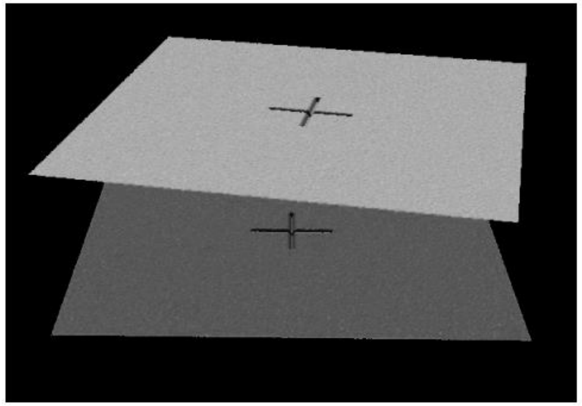


Image source: https://www.cs.cmu.edu/~halismai/cicp/

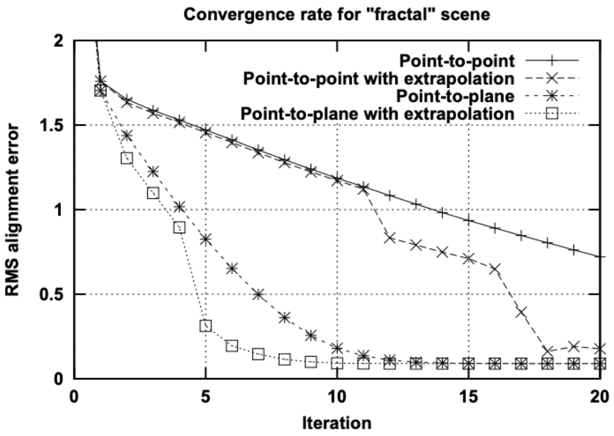
Point-to-Point vs Point-to-Plane

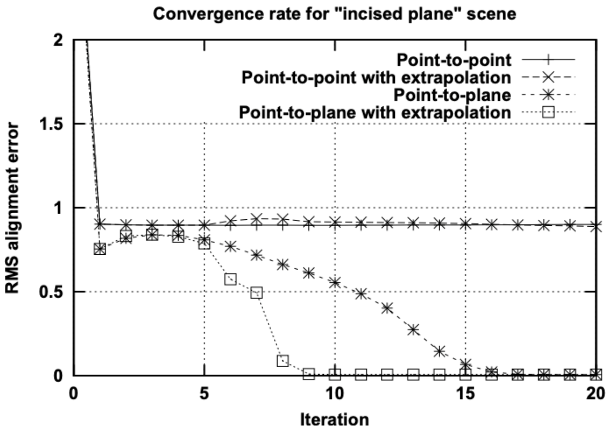


(b) Fractal landscape



(c) Incised plane





- Point-to-Point fails the ”incised plane” becase it doesn’t allow planes to slide.
- Usually point-to-plane is better.

Source: Efficient Variants of the ICP Algorithm, Szymon Rusinkiewicz, Marc Levoy

ICP Summary

1. Given two point sets
   a) Random sample / Voxel grid / NSS / Feature detector
2. Data association
   a) Nearest neighbor / Normal shooting / Projection / Feature matching
   b) Reject outliers
3. Compute R, t
   a) Point-to-Point
   b) Point-to-Plane
4. Check converge
   a) Cost / $\Delta R, \Delta t$ small enough → stop
   b) Else, apply R, t to the source points, repeat

ICP Summary

- Advantages
  - Simple
  - Works well given proper initialization
- Disadvantages
  - Requires good initialize R, t guess
  - Data association is not perfect
  - Nearest neighbor search can be slow
    - Acceleration by kd-tree / octree

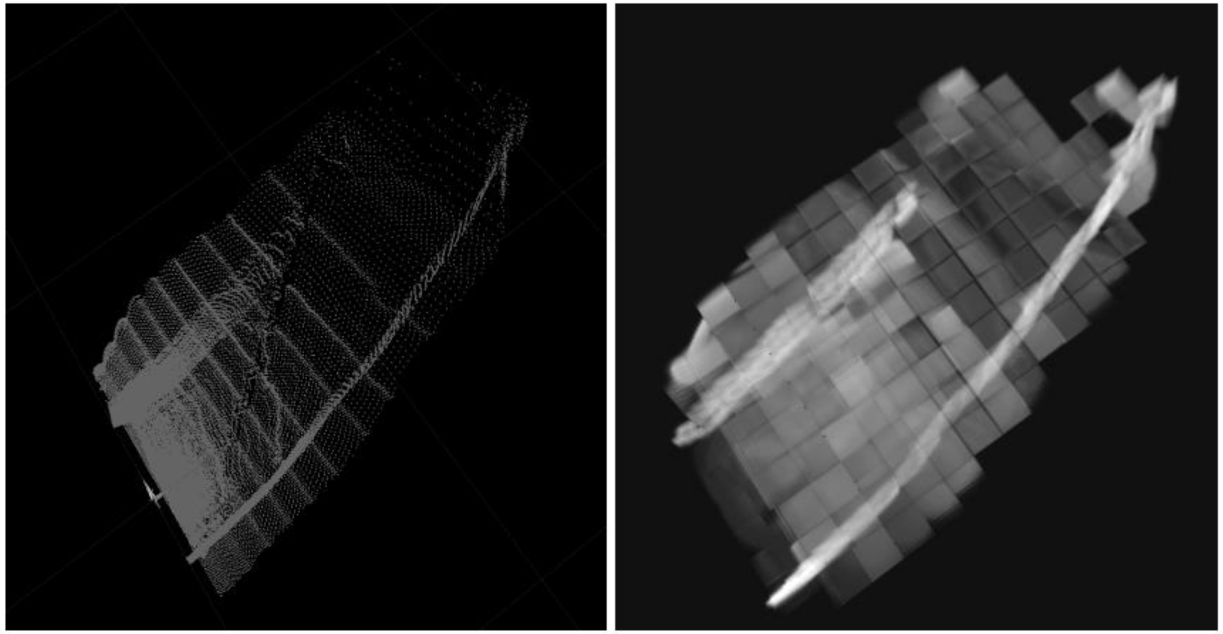
- ICP
  - Each point is considered without neighborhood info.
    - How about representing the neighborhood by probability?
  - Requires compute heavy association like nearest neighbor search
    - Can we avoid it?

- Yes, Normal Distribution Transform (NDT)

The Normal Distributions Transform: A New Approach to Laser Scan Matching, Peter Biber, Wolfgang Straber, IROS 2003

- Want neighborhood info?
  - Voxel grid.
  - Each cell is a neighborhood.

- Don’t want nearest neighbor search?
  - Voxel grid.
  - Floor operation gives coordinates.



(a) Original point cloud. (b) NDT representation.

3D-NDT representation for a tunnel, see from above. Brighter, denser parts represent higher probabilities.

Source: The Three-Dimensional Normal-Distributions Transform – an Efficient Representation for Registration, Surface Analysis, and Loo Detection, Martin Magnusson

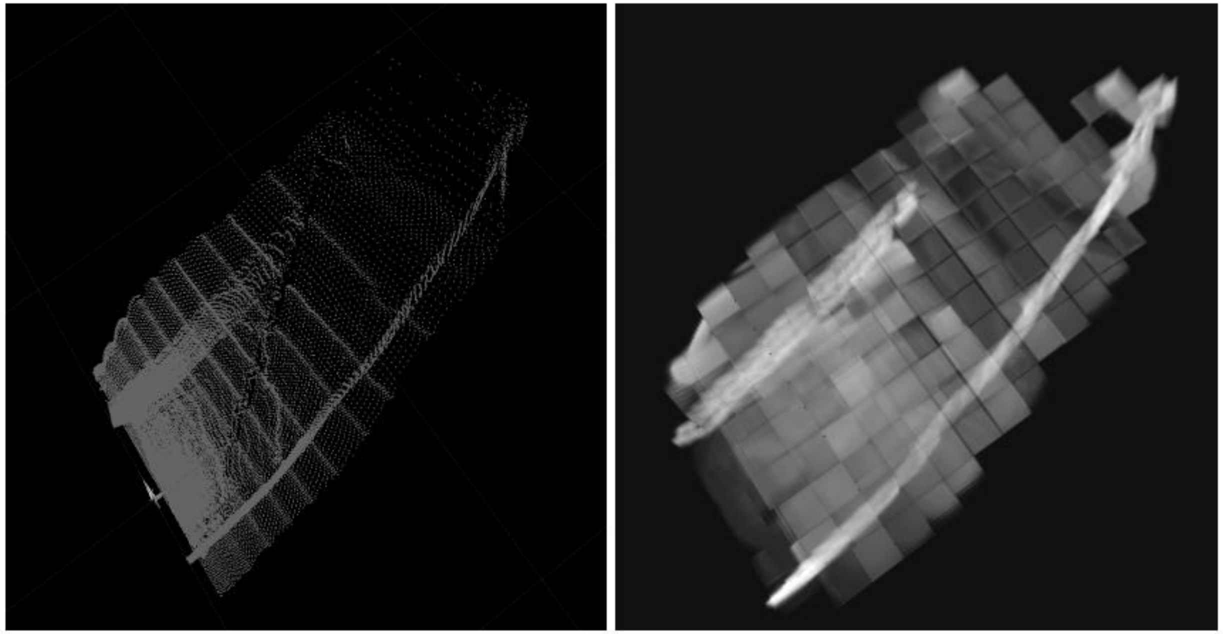
NDT – Representation

- Build a voxel grid over the target/destination point cloud.
- For cells with more than $m = 5$ points, compute

$$\vec{\mu} = \frac{1}{m} \sum_{k=1}^{m} \vec{y}_k,$$

$$\boldsymbol{\Sigma} = \frac{1}{m-1} \sum_{k=1}^{m} (\vec{y}_k - \vec{\mu})(\vec{y}_k - \vec{\mu})^{\mathrm{T}},$$

- $\{\vec{y}_k, k = 1, \cdots, m\}$ are points contained in a cell.



(a) Original point cloud. (b) NDT representation.

3D-NDT representation for a tunnel, see from above. Brighter, denser parts represent higher probabilities.

Source: The Three-Dimensional Normal-Distributions Transform – an Efficient Representation for Registration, Surface Analysis, and Loo Detection, Martin Magnusson

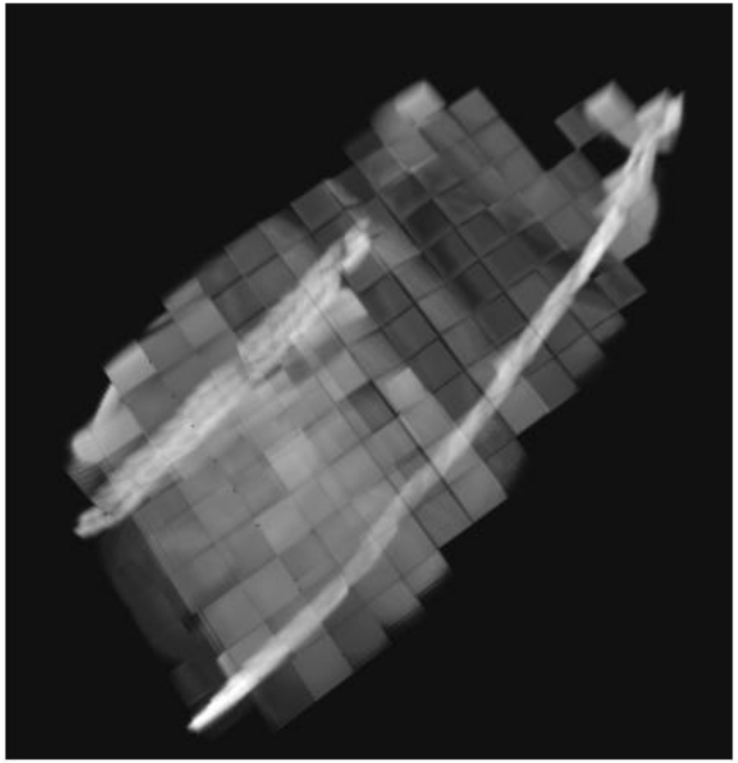
NDT – Representation

- For any point $\vec{x}$, find its cell
- That cell contains a Gaussian distribution

$$p(\vec{x}) = \frac{1}{(2\pi)^{D/2}\sqrt{|\boldsymbol{\Sigma}|}} \exp\left(-\frac{(\vec{x}-\vec{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\vec{x}-\vec{\mu})}{2} \right)$$

(a) Spherical: All eigenvalues approximately equal. (b) Linear: One eigenvalue much larger than the other two. (c) Planar: One eigenvalue much smaller than the others.

Different 3D Gaussian distribution with different covariance matrix



(b) NDT representation.

3D-NDT representation for a tunnel, see from above. Brighter, denser parts represent higher probabilities.

NDT – MLE (Maximum Likelihood Estimation)

- Consider point $\vec{x}_k \in \mathbb{R}^3$ in source point cloud
  - Apply transformation $\vec{p}$ to bring it to target frame
  - Unknown parameters $\vec{p} = \vec{p}_6 = \left[t_x, t_y, t_z, \phi_x, \phi_y, \phi_z\right]^T \in \mathbb{R}^6$
  - $\vec{x}_k' = T(\vec{p}, \vec{x}_k)$
- Likelihood of $\vec{x}_k'$ matching the target point cloud: $p(\vec{x}_k') = p\big(T(\vec{p}, \vec{x}_k)\big)$
- Likelihood for $k = 1, \cdots n$

$$\Psi = \prod_{k=1}^{n} p(T(\vec{p}, \vec{x}_k))$$

- Maximize likelihood → minimize negative log-likelihood

$$-\log \Psi = -\sum_{k=1}^{n} \log \left(p(T(\vec{p}, \vec{x}_k))\right)$$

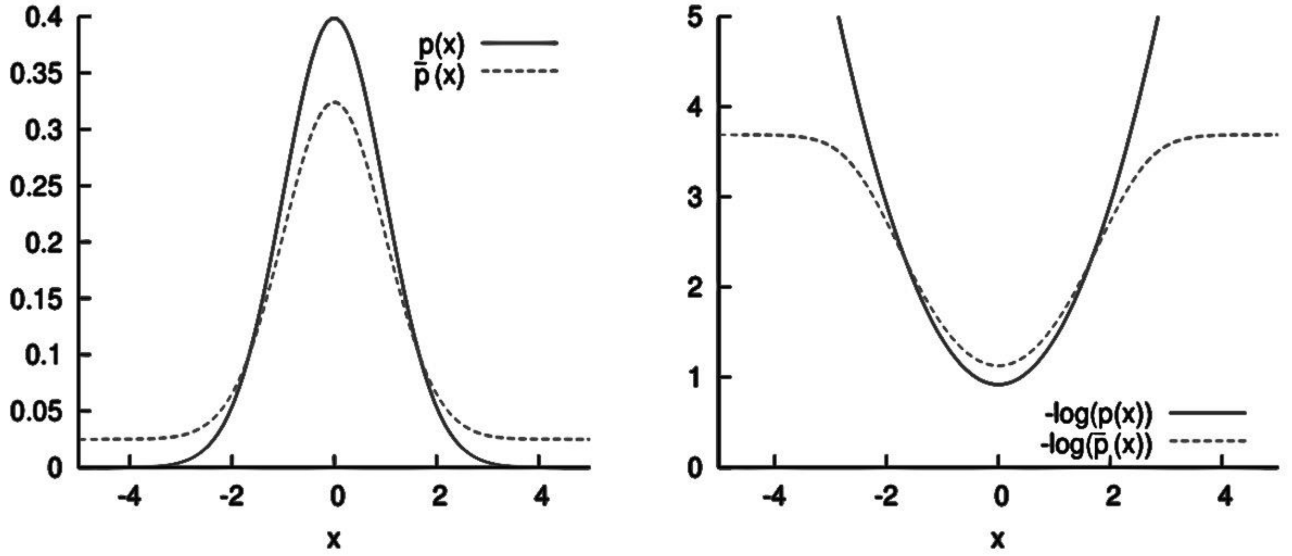
- NDT is a minimization

$$-\log \Psi = -\sum_{k=1}^{n} \log \left(p(T(\vec{p}, \vec{x}_k)) \right)$$

- Before we optimize it, two problems
  - Outlier → solved by mixture probability
  - Derivative is difficult → negative log approximation with Gaussian function

- Minimize negative log-likelihood is problematic for outliers
  - A outlier $\vec{x}_k \rightarrow (p \approx 0) \rightarrow (-\log p \approx \infty)$

$$-\log \Psi = -\sum_{k=1}^{n} \log \left(p(T(\vec{p}, \vec{x}_k))\right)$$



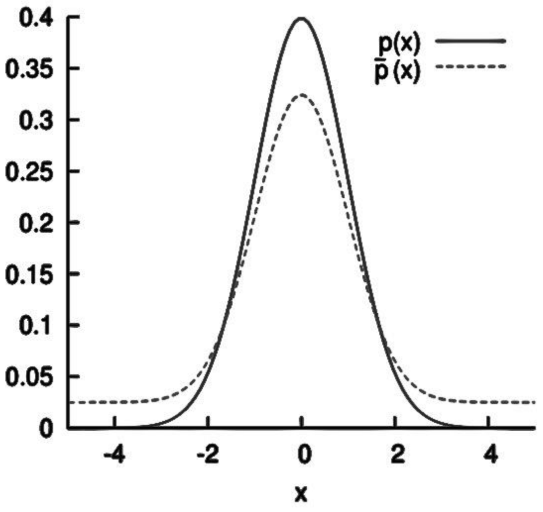
(a) Likelihood (b) Negative log-likelihood

Source: The Three-Dimensional Normal-Distributions Transform – an Efficient Representation for Registration, Surface Analysis, and Loo Detection, Martin Magnusson

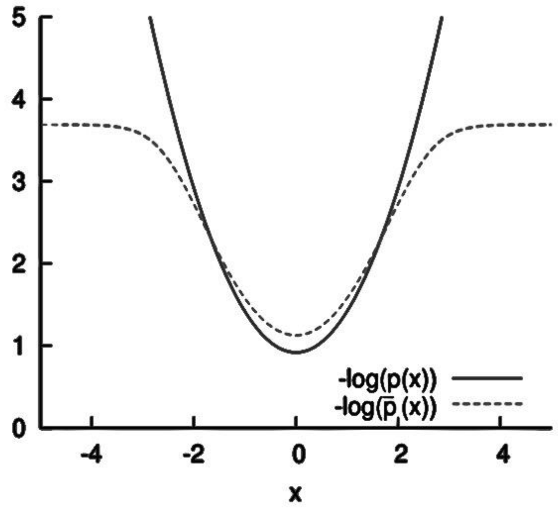
- How to avoid infinity?
  - Avoid likelihood p going to zero
  - Modify the probability to mixture of Gaussian and uniform distribution

$$\bar{p}(\vec{x}) = c_1 \exp\left(-\frac{(\vec{x}-\vec{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\vec{x}-\vec{\mu})}{2}\right) + c_2 \boxed{p_o}$$

Expected ratio of outliers



(a) Likelihood



(b) Negative log-likelihood

Source: The Three-Dimensional Normal-Distributions Transform – an Efficient Representation for Registration, Surface Analysis, and Loo Detection, Martin Magnusson

- How to solve c_1, c_2?

$$\bar{p}(\vec{x}) = c_1 \exp\left(-\frac{(\vec{x}-\vec{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\vec{x}-\vec{\mu})}{2}\right) + c_2 p_o$$

- Method 1
  - Cumulative Density Function (CDF) equals to one globally OR within the cell. c_1, c_2 are different per cell.

$$\iiint \bar{p}(\vec{x}) = 1 \qquad OR \qquad \iiint_{\vec{x}=\vec{x}_{min}}^{\vec{x}=\vec{x}_{max}} \bar{p}(\vec{x}) = 1$$

  - Not enough to constrain c_1, c_2
  - Add constraints like $c_1 + c_2 = 1$
- Method 2
  - Manually set c_1, c_2 for all cells (PCL's implementation). c_1, c_2 are the same for all cells.
  - $c_1 = 10 \cdot (1 - p_0)$
  - $c_2 = \frac{1}{resolution^3}$
  - Denote $c_2 p_0$ as the new c_2

NDT – Negative Log-likelihood Approximation

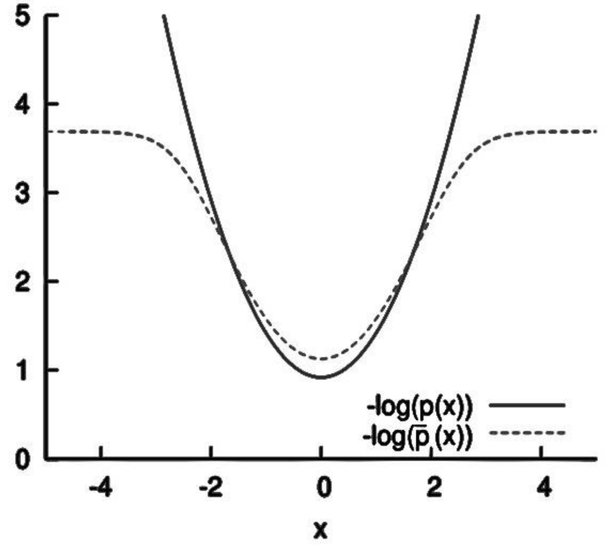
- Probability function of each cell

$$\bar{p}(\vec{x}) = c_1 \exp\left(-\frac{(\vec{x}-\vec{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\vec{x}-\vec{\mu})}{2}\right) + c_2$$

- The negative log-likelihood cost functions becomes

$$-\log\Psi = -\sum_{k=1}^{n} \log\left(\bar{p}(\vec{x}_k')\right) = -\sum_{k=1}^{n} \log\left(\bar{p}\left(T(\vec{p},\vec{x}_k)\right)\right)$$

- Derivative is difficult
  - Approximate $-\log \bar{p}(\vec{x})$ by a Gaussian function $\tilde{p}(\vec{x})$
  - $\tilde{p}(\vec{x}) = d_1 \exp\left(-\frac{d_2(\vec{x}-\vec{\mu})^T\Sigma^{-1}(\vec{x}-\vec{\mu})}{2}\right) + d_3$
  - How to get d_1, d_2, d_3?



(b) Negative log-likelihood

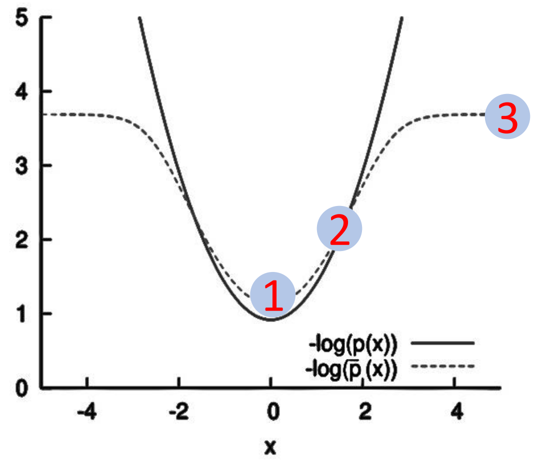
NDT – Negative Log-likelihood Approximation

$$-\log \bar{p}(\vec{x}) = -\log\left(c_1 \exp\left(-\frac{(\vec{x}-\vec{\mu})^T\Sigma^{-1}(\vec{x}-\vec{\mu})}{2}\right) + c_2\right)$$

$$\tilde{p}(\vec{x}) = d_1 \exp\left(-\frac{d_2(\vec{x}-\vec{\mu})^T\Sigma^{-1}(\vec{x}-\vec{\mu})}{2}\right) + d_3$$

Approximation $-\log \bar{p}(\vec{x}) \approx \tilde{p}(\vec{x})$		
Anchor point 1	$(\vec{x}-\vec{\mu}) = \vec{0}$	$-\log(c_1 + c_2) = d_1 + d_3$
Anchor point 2	$(\vec{x}-\vec{\mu})^T\Sigma^{-1}(\vec{x}-\vec{\mu}) = 1$	$-\log(c_1 e^{-0.5} + c_2) = d_1 e^{-0.5 d_2} + d_3$
Anchor point 3	$(\vec{x}-\vec{\mu}) = \vec{\infty}$	$-\log(c_2) = d_3$

$$d_3 = -\log(c_2),$$
$$d_1 = -\log(c_1 + c_2) - d_3,$$
$$d_2 = -2\log\left(\left(-\log\left(c_1 \exp(-1/2) + c_2\right) - d_3\right)/d_1\right)$$



- NDT is an minimization problem

$$-\log \Psi = -\sum_{k=1}^{n} \log\left(p(T(\vec{p}, \vec{x}_k))\right)$$

- Now it becomes minimization

$$s(\vec{p}) = \sum_{k=1}^{n} \tilde{p}\big(T(\vec{p}, \vec{x}_k)\big) = \sum_{k=1}^{n} d_1 \exp\left(-\frac{d_2 (T(\vec{p}, \vec{x}_k) - \vec{\mu}_k)^T \Sigma_\mathrm{k}^{-1} (T(\vec{p}, \vec{x}_k) - \vec{\mu}_k)}{2}\right) + d_3$$

- How to optimize? Iterative.
  - At step $\vec{p}_i$
  - Solve $\Delta\vec{p}$ to obtain $\vec{p}_{i+1} = \vec{p}_i + \Delta\vec{p}$

- Iterative optimization
  - At position x_i, find x_{i+1}

- Simple methods
  - First-order: Gradient descent
  - Second-order: Newton's method

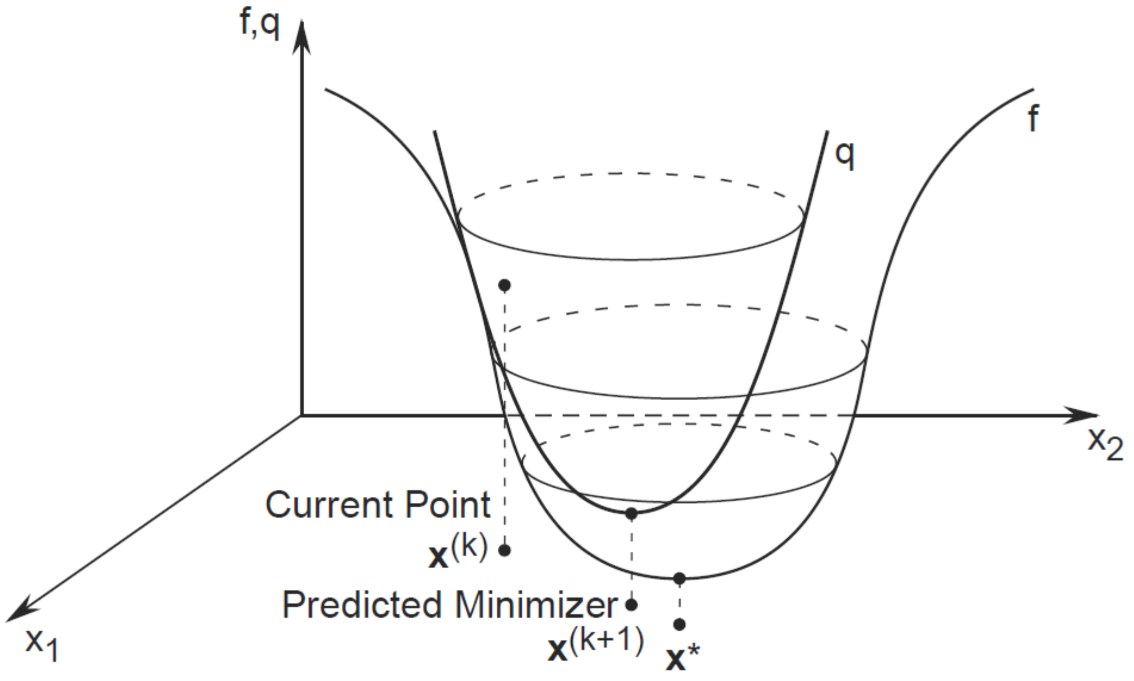


Image source: https://suzyahyah.github.io/calculus/optimization/2018/04/06/Taylor-Series-Newtons-Method.html

Newton's Method in Optimization

- Optimization over $f: \mathbb{R}^n \to \mathbb{R}$
- Taylor series approximation at position $x_i \in \mathbb{R}^n$

$$f(x) \approx f(x_i) + (x - x_i)^T \mathrm{g}_i + \frac{1}{2}(x - x_i)^T H_i (x - x_i)$$

  - Jacobian vector $\mathrm{g}_i = \left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_{x=x_i} \in \mathbb{R}^n$
  - Hessian matrix $H_i = \left.\frac{\mathrm{d}^2 f}{\mathrm{d}x\mathrm{d}x}\right|_{x=x_i} \in \mathbb{R}^{n \times n}$

- Compute first-order derivative, make it zero

$$0 = \mathrm{g}_i + H_i(x - x_i)$$

$$\Delta x = x - x_i = -H_i^{-1}\mathrm{g}_i$$

$$x_{i+1} = x_i - H_i^{-1}\mathrm{g}_i$$

NDT – MLE

- NDT is an minimization

$$s(\vec{p}) = \sum_{k=1}^{n} \tilde{p}\left(T(\vec{p}, \vec{x}_k)\right) = \sum_{k=1}^{n} d_1 \exp\left(-\frac{d_2 (T(\vec{p}, \vec{x}_k) - \vec{\mu}_k)^T \Sigma_\mathrm{k}^{-1} (T(\vec{p}, \vec{x}_k) - \vec{\mu}_k)}{2}\right) + d_3$$

$$s(\vec{p}) = \sum_{k=1}^{n} d_1 \exp\left(-\frac{d_2 \vec{x}_k'^{\,T} \Sigma_\mathrm{k}^{-1} \vec{x}_k'}{2}\right)$$

$\vec{x}_k' \equiv T(\vec{p}, \vec{x}_k) - \vec{\mu}_k, d_3$ is constant

- At each iteration, compute $\Delta\vec{p} = -H^{-1}\vec{\mathrm{g}}$
- Jacobian vector $\vec{\mathrm{g}}$ with elements denoted as g_i

$$g_i = \frac{\delta s}{\delta p_i} = \sum_{k=1}^{n} d_1 d_2 \vec{x}_k'^{\mathrm{T}} \mathbf{\Sigma}_k^{-1} \frac{\delta \vec{x}_k'}{\delta p_i} \exp\left(\frac{-d_2}{2} \vec{x}_k'^{\mathrm{T}} \mathbf{\Sigma}_k^{-1} \vec{x}_k'\right)$$

- Hessian matrix H with elements denoted as H_{ij}

$$H_{ij} = \frac{\delta^2 s}{\delta p_i \delta p_j} = \sum_{k=1}^{n} d_1 d_2 \exp\left(\frac{-d_2}{2} \vec{x}_k'^{\mathrm{T}} \mathbf{\Sigma}_k^{-1} \vec{x}_k'\right) \left(-d_2 \left(\vec{x}_k'^{\mathrm{T}} \mathbf{\Sigma}_k^{-1} \frac{\delta \vec{x}_k'}{\delta p_i}\right) \left(\vec{x}_k'^{\mathrm{T}} \mathbf{\Sigma}_k^{-1} \frac{\delta \vec{x}_k'}{\delta p_j}\right) + \vec{x}_k'^{\mathrm{T}} \mathbf{\Sigma}_k^{-1} \frac{\delta^2 \vec{x}_k'}{\delta p_i \delta p_j} + \frac{\delta \vec{x}_k'}{\delta p_j}^{\mathrm{T}} \mathbf{\Sigma}_k^{-1} \frac{\delta \vec{x}_k'}{\delta p_i}\right)$$

NDT

- Build voxel grid
- Compute probability function for each cell
  - $\vec{\mu}, \vec{\Sigma}$ are different per-cell
  - c_1, c_2, p_0 are the same for all cells
  - d_1, d_2, d_3 are the same for all cells.

- Each NDT iteration
  - Perform newton's method once
  - Update the parameters $\vec{p}$
  - Transform source points χ

Source: The Three-Dimensional Normal-Distributions Transform – an Efficient Representation for Registration, Surface Analysis, and Loo Detection, Martin Magnusson

Algorithm 2 Register scan $\mathcal{X}$ to reference scan $\mathcal{Y}$ using NDT.

ndt($\mathcal{X}, \mathcal{Y}, \vec{p}$)

{Initialisation:}
allocate cell structure $\mathcal{B}$
for all points $\vec{y}_k \in \mathcal{Y}$ **do**
  find the cell $b_i \in \mathcal{B}$ that contains $\vec{y}_k$
  store $\vec{y}_k$ in b_i
end for
for all cells $b_i \in \mathcal{B}$ **do**
  $\mathcal{Y}' = \{\vec{y}'_1, \ldots, \vec{y}'_m\} \leftarrow$ all points in b_i
  $\vec{\mu}_i \leftarrow \frac{1}{n} \sum_{k=1}^{m} \vec{y}'_k$
  $\Sigma_i \leftarrow \frac{1}{m-1} \sum_{k=1}^{m} (\vec{y}'_k - \vec{\mu})(\vec{y}'_k - \vec{\mu})^{\mathrm{T}}$
end for
{Registration:}
while not converged **do**
  score $\leftarrow 0$
  $\vec{g} \leftarrow 0$
  $\mathbf{H} \leftarrow 0$
  for all points $\vec{x}_k \in \mathcal{X}$ **do**
    find the cell b_i that contains $T(\vec{p}, \vec{x}_k)$
    score $\leftarrow$ *score* $+ \tilde{p}\left(T(\vec{p}, \vec{x}_k)\right)$ (see Equation 6.9)
    update $\vec{g}$ (see Equation 6.12)
    update $\mathbf{H}$ (see Equation 6.13)
  end for
  solve $\mathbf{H}\Delta\vec{p} = -\vec{g}$
  $\vec{p} \leftarrow \vec{p} + \Delta\vec{p}$
end while

No need to perform NN search

The two equations on the last page.

- Jacobian vector $\vec{g}$ with elements denoted as g_i

$$\vec{x}' \equiv T(\vec{p}, \vec{x}_k) - \vec{\mu}_k$$

$$g_i = \frac{\delta s}{\delta p_i} = \sum_{k=1}^{n} d_1 d_2 \vec{x}_k'^{\mathrm{T}} \boldsymbol{\Sigma}_k^{-1} \boxed{\frac{\delta \vec{x}_k'}{\delta p_i}} \exp\left(\frac{-d_2}{2} \vec{x}_k'^{\mathrm{T}} \boldsymbol{\Sigma}_k^{-1} \vec{x}_k'\right)$$

- Hessian matrix H with elements denoted as H_{ij}

$$H_{ij} = \frac{\delta^2 s}{\delta p_i \delta p_j} = \sum_{k=1}^{n} d_1 d_2 \exp\left(\frac{-d_2}{2} \vec{x}_k'^{\mathrm{T}} \boldsymbol{\Sigma}_k^{-1} \vec{x}_k'\right) \left(-d_2 \left(\vec{x}_k'^{\mathrm{T}} \boldsymbol{\Sigma}_k^{-1} \frac{\delta \vec{x}_k'}{\delta p_i}\right) \left(\vec{x}_k'^{\mathrm{T}} \boldsymbol{\Sigma}_k^{-1} \frac{\delta \vec{x}_k'}{\delta p_j}\right) + \vec{x}_k'^{\mathrm{T}} \boldsymbol{\Sigma}_k^{-1} \frac{\delta^2 \vec{x}_k'}{\delta p_i \delta p_j} + \frac{\delta \vec{x}_k'}{\delta p_j}^{\mathrm{T}} \boldsymbol{\Sigma}_k^{-1} \frac{\delta \vec{x}_k'}{\delta p_i}\right)$$

- Unknown parameters $\vec{p} = \vec{p}_6 = \left[t_x, t_y, t_z, \phi_x, \phi_y, \phi_z\right]^T \in \mathbb{R}^6$
- Transform can be represented *as* $T_E(\vec{p}_6, \vec{x}) = R_{\phi_x \phi_y \phi_z} \vec{x} + \vec{t}$
  - Similar to ICP Point-to-Plane formulation

Recall – ICP Point-to-Plane Rotation

- Represent $R \in \mathbb{R}^3$ by angles
  - x axis by α, y axis by β, z axis by γ

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$$

$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}$$

$$R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R = R_z(\gamma)R_y(\beta)R_x(\alpha) = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

$$\begin{aligned}
r_{11} &= \cos\gamma\cos\beta, \\
r_{12} &= -\sin\gamma\cos\alpha + \cos\gamma\sin\beta\sin\alpha, \\
r_{13} &= \sin\gamma\sin\alpha + \cos\gamma\sin\beta\cos\alpha, \\
r_{21} &= \sin\gamma\cos\beta, \\
r_{22} &= \cos\gamma\cos\alpha + \sin\gamma\sin\beta\sin\alpha, \\
r_{23} &= -\cos\gamma\sin\alpha + \sin\gamma\sin\beta\cos\alpha, \\
r_{31} &= -\sin\beta, \\
r_{32} &= \cos\beta\sin\alpha, \\
r_{33} &= \cos\beta\cos\alpha.
\end{aligned}$$

- Transformation for 3D NDT. $c_i = \cos\phi_i\,, s_i = \sin\phi_i$

$$T_E(\vec{p}_6, \vec{x}) = \mathbf{R}_x\mathbf{R}_y\mathbf{R}_z\vec{x} + \vec{t} = \begin{bmatrix} c_y c_z & -c_y s_z & s_y \\ c_x s_z + s_x s_y c_z & c_x c_z - s_x s_y s_z & -s_x c_y \\ s_x s_z - c_x s_y c_z & c_x s_y s_z + s_x c_z & c_x c_y \end{bmatrix} \vec{x} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

- Jacobian of the transformation $J_E = \left[\frac{\partial T_E(\vec{p}_6,\vec{x})}{\partial p_1}, \cdots, \frac{\partial T_E(\vec{p}_6,\vec{x})}{\partial p_6}\right] \in \mathbb{R}^{3\times 6}$

$$\begin{aligned}
a &= x_1(-s_x s_z + c_x s_y c_z) + x_2(-s_x c_z - c_x s_y s_z) + x_3(-c_x c_y),\\
b &= x_1(c_x s_z + s_x s_y c_z) + x_2(-s_x s_y s_z + c_x c_z) + x_3(-s_x c_y),\\
c &= x_1(-s_y c_z) + x_2(s_y s_z) + x_3(c_y),\\
d &= x_1(s_x c_y c_z) + x_2(-s_x c_y s_z) + x_3(s_x s_y),\\
e &= x_1(-c_x c_y c_z) + x_2(c_x c_y s_z) + x_3(-c_x s_y),\\
f &= x_1(-c_y s_z) + x_2(-c_y c_z),\\
g &= x_1(c_x c_z - s_x s_y s_z) + x_2(-c_x s_z - s_x s_y c_z),\\
h &= x_1(s_x c_z + c_x s_y s_z) + x_2(c_x s_y c_z - s_x s_z).
\end{aligned}$$

Numerator layout

$$\mathbf{J}_E = \begin{bmatrix} 1 & 0 & 0 & 0 & c & f \\ 0 & 1 & 0 & a & d & g \\ 0 & 0 & 1 & b & e & h \end{bmatrix}$$

NDT – Jacobian / Hessian for 3D NDT

- Transformation for 3D NDT. $c_i = \cos\phi_i, s_i = \sin\phi_i$

$$T_E(\vec{p}_6, \vec{x}) = \mathbf{R}_x\mathbf{R}_y\mathbf{R}_z\vec{x} + \vec{t} = \begin{bmatrix} c_y c_z & -c_y s_z & s_y \\ c_x s_z + s_x s_y c_z & c_x c_z - s_x s_y s_z & -s_x c_y \\ s_x s_z - c_x s_y c_z & c_x s_y s_z + s_x c_z & c_x c_y \end{bmatrix} \vec{x} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

- Hessian of the transformation $H_E, \vec{H}_{ij} = \frac{\partial J_j}{\partial p_i} = \frac{\partial^2 T_E(\vec{p}_6, \vec{x})}{\partial p_i \partial p_j} \in \mathbb{R}^3$

$$J_E = [J_1, J_2, J_3, J_4, J_5, J_6] = \begin{bmatrix} 1 & 0 & 0 & 0 & c & f \\ 0 & 1 & 0 & a & d & g \\ 0 & 0 & 1 & b & e & h \end{bmatrix}$$

$$H_E = \begin{bmatrix} \frac{\partial J_1}{\partial p_1} & \frac{\partial J_2}{\partial p_1} & \cdots & \frac{\partial J_6}{\partial p_1} \\ \frac{\partial J_1}{\partial p_2} & \frac{\partial J_2}{\partial p_2} & \cdots & \frac{\partial J_6}{\partial p_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial J_1}{\partial p_6} & \frac{\partial J_2}{\partial p_6} & \cdots & \frac{\partial J_6}{\partial p_6} \end{bmatrix}$$

- Hessian of the transformation $H_E, \vec{H}_{ij} = \frac{\partial J_j}{\partial p_i} = \frac{\partial^2 T_E(\vec{p}_6,\vec{x})}{\partial p_i \partial p_j} \in \mathbb{R}^3$

$$J_E = [J_1, J_2, J_3, J_4, J_5, J_6] = \begin{bmatrix} 1 & 0 & 0 & 0 & c & f \\ 0 & 1 & 0 & a & d & g \\ 0 & 0 & 1 & b & e & h \end{bmatrix}$$

$$H_E = \begin{bmatrix} \frac{\partial J_1}{\partial p_1} & \frac{\partial J_2}{\partial p_1} & \cdots & \frac{\partial J_6}{\partial p_1} \\ \frac{\partial J_1}{\partial p_2} & \frac{\partial J_2}{\partial p_2} & \cdots & \frac{\partial J_6}{\partial p_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial J_1}{\partial p_6} & \frac{\partial J_2}{\partial p_6} & \cdots & \frac{\partial J_6}{\partial p_6} \end{bmatrix}$$

$$\mathbf{H}_E = \begin{bmatrix} \vec{H}_{11} & \cdots & \vec{H}_{16} \\ \vdots & \ddots & \vdots \\ \vec{H}_{61} & \cdots & \vec{H}_{66} \end{bmatrix} = \begin{bmatrix} \vec{0} & \vec{0} & \vec{0} & \vec{0} & \vec{0} & \vec{0} \\ \vec{0} & \vec{0} & \vec{0} & \vec{0} & \vec{0} & \vec{0} \\ \vec{0} & \vec{0} & \vec{0} & \vec{0} & \vec{0} & \vec{0} \\ \vec{0} & \vec{0} & \vec{0} & \vec{a} & \vec{b} & \vec{c} \\ \vec{0} & \vec{0} & \vec{0} & \vec{b} & \vec{d} & \vec{e} \\ \vec{0} & \vec{0} & \vec{0} & \vec{c} & \vec{e} & \vec{f} \end{bmatrix}$$

$$\vec{a} = \begin{bmatrix} 0 \\ x_1(-c_x s_z - s_x s_y c_z) + x_2(-c_x c_z + s_x s_y s_z) + x_3(s_x c_y) \\ x_1(-s_x s_z + c_x s_y c_z) + x_2(-c_x s_y s_z - s_x c_z) + x_3(-c_x c_y) \end{bmatrix}, \quad \vec{d} = \begin{bmatrix} x_1(-c_y c_z) + x_2(c_y s_z) + x_3(-s_y) \\ x_1(-s_x s_y c_z) + x_2(s_x s_y s_z) + x_3(s_x c_y) \\ x_1(c_x s_y c_z) + x_2(-c_x s_y s_z) + x_3(-c_x c_y) \end{bmatrix},$$

$$\vec{b} = \begin{bmatrix} 0 \\ x_1(c_x c_y c_z) + x_2(-c_x c_y s_z) + x_3(c_x s_y) \\ x_1(s_x c_y c_z) + x_2(-s_x c_y s_z) + x_3(s_x s_y) \end{bmatrix}, \quad \vec{e} = \begin{bmatrix} x_1(s_y s_z) + x_2(s_y c_z) \\ x_1(-s_x c_y s_z) + x_2(-s_x c_y c_z) \\ x_1(c_x c_y s_z) + x_2(c_x c_y c_z) \end{bmatrix},$$

$$\vec{c} = \begin{bmatrix} 0 \\ x_1(-s_x c_z - c_x s_y s_z) + x_2(-s_x s_z - c_x s_y c_z) \\ x_1(c_x c_z - s_x s_y s_z) + x_2(-s_x s_y c_z - c_x s_z) \end{bmatrix}, \quad \vec{f} = \begin{bmatrix} x_1(-c_y c_z) + x_2(c_y s_z) \\ x_1(-c_x s_z - s_x s_y c_z) + x_2(-c_x c_z + s_x s_y s_z) \\ x_1(-s_x s_z + c_x s_y c_z) + x_2(-c_x s_y s_z - s_x c_z) \end{bmatrix}.$$

- So complicated → Simplify as $\sin\phi \approx \phi, \cos\phi \approx 1, \phi^2 \approx 0$ when $\phi \approx 0$
- Transformation becomes

$$T_E(\vec{p}_6, \vec{x}) = \begin{bmatrix} c_y c_z & -c_y s_z & s_y \\ c_x s_z + s_x s_y c_z & c_x c_z - s_x s_y s_z & -s_x c_y \\ s_x s_z - c_x s_y c_z & c_x s_y s_z + s_x c_z & c_x c_y \end{bmatrix} \vec{x} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \approx \begin{bmatrix} 1 & -\phi_z & \phi_y \\ \phi_z & 1 & -\phi_x \\ -\phi_y & \phi_x & 1 \end{bmatrix} \vec{x} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

- Jacobian is simply $\tilde{\mathbf{J}}_E = \begin{bmatrix} 1 & 0 & 0 & 0 & x_3 & -x_2 \\ 0 & 1 & 0 & -x_3 & 0 & x_1 \\ 0 & 0 & 1 & x_2 & -x_1 & 0 \end{bmatrix}$

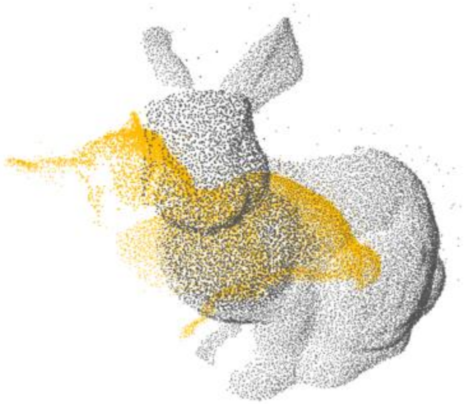
- Hessian matrix is simply $\widetilde{H}_E = \mathbf{0} \in \mathbb{R}^{18\times 6}$

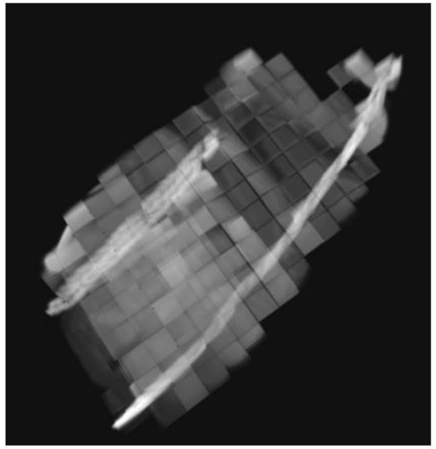
1. Build voxel grid for target points
    1. Compute μ, Σ for each cell
    2. Compute d_1, d_2, d_3 as constant
2. Initialize the parameters $\vec{p}$
3. Iterate
    1. Transform source points by $\vec{p}$
    2. Compute cost, Jacobian, Hessian
    3. Update $\vec{p}$ by Newton's method

- Advantages
    - No nearest neighbor search → faster
    - Less sensitive to initialization compared with ICP
- Disadvantages
    - More complicated procedure
    - Need parameter tuning of voxel grid resolution

ICP vs NDT

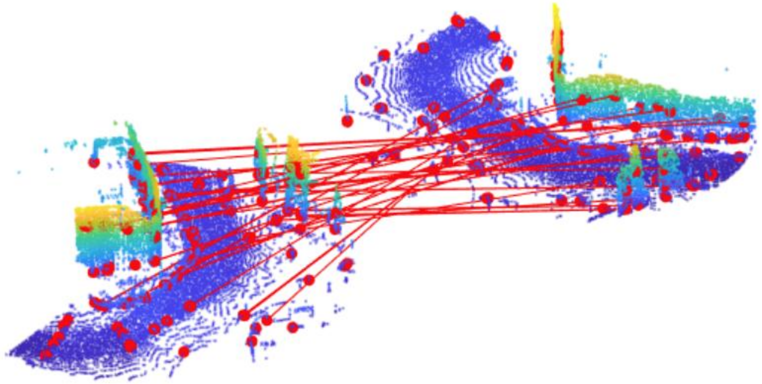
	ICP	NDT
Pose Initialization	Yes	Yes
Target Point Cloud Voxel Grid	No	Yes
Nearest Neighbor Search	Yes	No
Optimization Problem	Procrustes Transformation	Maximum-likelihood
Closed-Form Solution at each Iteration	Yes	No





- What if there isn't proper initial guess for the pose?
- Feature detection + description + RANSAC(Random Sample Consensus)

1. Feature detection & description on source and target point cloud.
2. Establish correspondences (point pairs)
3. RANSAC iterations
    1. Select 3 pairs at each iteration
    2. Solve R, t by Procrustes Transformation
    3. Compute number of inlier pairs
4. Select R, t with most inliers



Source: USIP: Unsupervised Stable Interest Point Detection from 3D Point Clouds, Jiaxin Li and Gim Hee Lee

- Input:
  - Source keypoints & descriptors
  - Target keypoints & descriptors
- Methods to establish correspondences between two point clouds
  - Similar to the nearest neighbor similarity graph in spectral clustering
  - 1. Nearest descriptor matching (3 methods)
    a) For each source keypoint s_i, find a target keypoint t_i with most similar descriptor (L2-norm)
    b) For each target keypoint t_i, find a source keypoint s_i with most similar descriptor (L2-norm)
    c) Combination of the above
  - 2. Mutual nearest descriptor matching
    - Build a pair only if the following holds
      - s_i is the nearest neighbor of t_i (in descriptor space)
      - t_i is the nearest neighbor of s_i (in descriptor space)

Registration Pipeline

1. Data pre-processing
    1. Downsample
    2. Noise removal
2. Determine initial pose
    1. Prior information
    2. Other information like odometry, IMU (Inertial Measurement Unit), etc.
    3. Feature detection + description + matching + RANSAC
3. Run registration algorithms
    1. ICP
    2. NDT
    3. Others like grid based optimization.

- Implement feature detectors & descriptors
  - Any algorithm you want
  - You may call APIs. But still, your own implementation is preferred.
- Implement your own ICP or NDT.
  - Do NOT call APIs except for nearest neighbor search.
- Test your registration algorithm on the provided dataset
  - There is NO proper initialization provided.
  - Report the following metrics. Evaluation script is provided.
    - Average Relative Rotational Error (RRE)
    - Average Relative Translational Error (RTE)
    - Percentage of successful registration

Homework

- We provide the registration dataset that contains 342 pairs of point clouds.
- You are required to provide your registration results into “reg_result.txt”
  - The original “reg_result.txt” is an example with 3 ground truth results.
  - The rest of 339 ground truth results are not provided.
- There is the “evaluate_rt.py”, it provides
  - Functions to read and visualize the pairs
  - Functions to evaluate the RRE, RTE, success rate