




# ICP-SLAM as LS Optimization problem

☰ Comments	
# Order (according to me)	
▼ STATUS	
☰ Week (according to Prof Madhav's original email)?	


## ▼ 22 Sep Class — Real-time updates: (Prof is currently at )

- 4:00 pm Localization and Mapping Jacobians
- 4:10 pm Overall structure of Jacobian
- 4:20 pm Typical NLS update
- 4:25 pm Localization update
-  4:35 pm Dimensions of Jacobian
- 4:40 pm Started next lecture: Loop Closure

If you're a student, please note that there are some mistakes on this page, so proceed with caution. We're rectifying these ASAP and will notify on Moodle once it is done. Status—

STATUS: 95% done

## Prof's notes on ICP-SLAM

 Prof Madhav's notes on ICP-SLAM

(This Notion page link if you're viewing a PDF)

Prof's notes on ICP-SLAM

## 0. Need for SLAM Backend or Multiview ICP "Optimization"

### Key Insight

Why is this  $\hat{X}_{ij}$  different from the observations  $X_{ij}$ ?

Various ways to approach this "multiview ICP"

### 1. ICP-SLAM as Optimization

Num of variables

Update step

Update step for  $\mathbf{T}$

Obtaining the matrix  $\mathbf{T}$  from the vector  $\xi$  through exp map

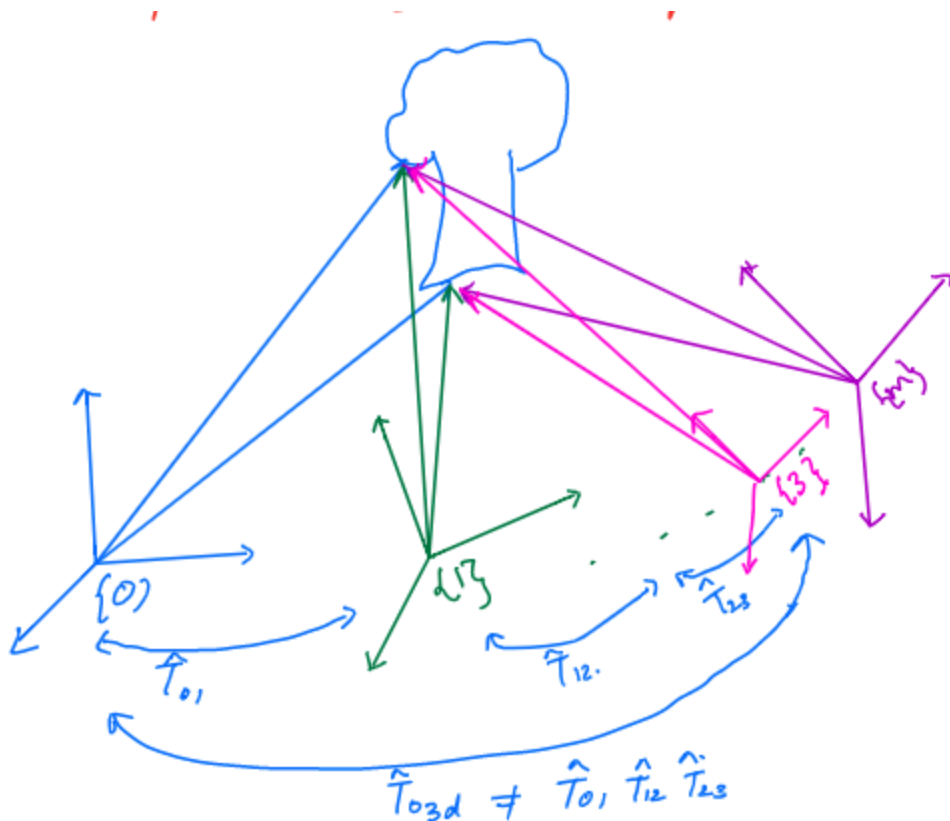
Obtaining the vector  $\xi$  from the matrix  $\mathbf{T}$

Moving forward with calculating Jacobian



Summarizing the story so far...

Revisiting our residual error function & update

## 0. Need for SLAM Backend or Multiview ICP "Optimization"



In the figure:

- Blue frame  $\{0\}$   —  $T_0 = [I|0]$  is the origin.
- Blue tree  —  $X_{0j} = \begin{bmatrix} x_{0j} & y_{0j} & z_{0j} \end{bmatrix}^T$ : where  $j = \{1, \dots, n\}$  are the coordinates of  $n$  points in frame  $\{0\}$  obtained from depth, azimuth and elevation measurements.
  - Since the depth for every point  $j$  in frame  $\{0\}$  is in error, hence  $X_{0j}$  is also in error.
  - So is the case for  $X_{1j}$  (frame 1) and hence, relative pose estimates between successive frames  $\{i, i + 1\}$  i.e.  $\hat{T}_{01}, \hat{T}_{12}, \dots, \hat{T}_{(m-1), m}$  are all in error.
- How to alleviate this error?
  - Filtering methods: Last topic of the semester (After Vision)
  - Optimization methods: Now — Pose this as **"multiview optimization"**.

## Key Insight

- If there is a frame  $\{q\}$  in which the depth measurements and the point cloud  $X_{qj}$  are particularly noiseless: Can this be used to alleviate other views in terms of poses and 3D points estimated in those views
- If a set of  $n$  points are viewed in  $m$  frames or observations, what is the best estimate for these  $n$  points and  $m$  poses.
  - **Multiview Aggregation**
  - **Multiview Consistency**
- Let the points  $j = \{1, \dots, n\}$  be represented in frames  $i = \{1, \dots, m\}$  as  $X_{ij} = \begin{bmatrix} x_{ij} & y_{ij} & z_{ij} \end{bmatrix}^T$  as  $mn$  observations.  **$\hat{X}_{ij}$  are estimated from ICP while  $X_{ij}$  are obtained directly through a sensor say LiDAR.**
- Also based on the observation of  $j$  in  $\{0\}$  which is  $X_{0j}$  and  $\hat{T}_{0i}$  **estimated from ICP** as  $\hat{T}_{0i} = \hat{T}_{01} \hat{T}_{12} \dots \hat{T}_{(i-1), i}$  we predict what is  $X_{0j}$  in frame  $i$  as

$$\hat{X}_{ij} = \hat{T}_{i0} X_{0j} \quad (0.1)$$

## Why is this $\hat{X}_{ij}$ different from the observations $X_{ij}$ ?

- Two reasons! Participate by commenting! (for students)
  1. **TODO** through participation
  2. **TODO** through participation

## Various ways to approach this "multiview ICP"

We go with the first procedure below.

1. When I aggregate the same  $n$  points from multiple views

$\hat{X}_{0j}^i = \hat{T}_{0i} X_{ij}$ , I get  $m$  sets of  $n$  points in frame  $\{0\}$  that I average as

$$\hat{X}_{0j} = \sum_{i=1}^m \frac{\hat{X}_{0j}^i}{m} \quad (0.2)$$

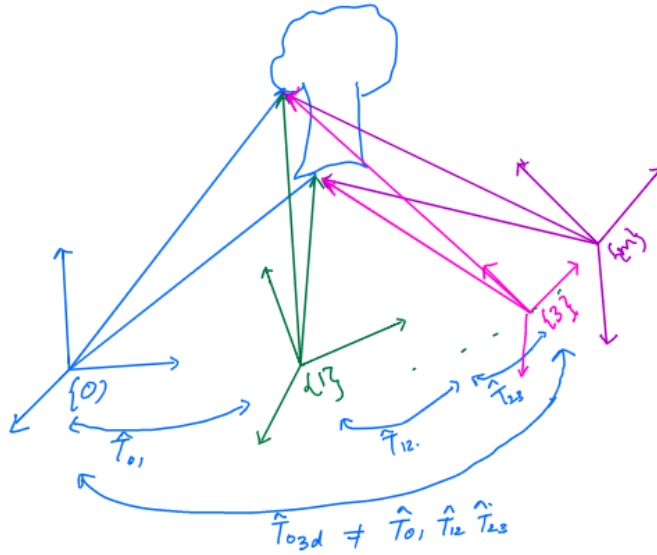
or such aggregation over the  $m$  frames.

2. I can also use an ICP estimate directly between the points in frames  $\{0\}$  and  $\{i\}$  and get a  $\hat{T}_{0i}$  and say that the 2 sets of points should be the same which is

$$\hat{X}_{ij} = \hat{T}_{0i} \hat{X}_{0j} \quad (0.3)$$

should be the same as  $\hat{X}_{ij}$  predicted/estimated by (0.1). The difference is that here, we are estimating ICP directly while above, we are estimating ICP between immediate pair of point clouds.

## 1. ICP-SLAM as Optimization



$\vec{X}$  (represented by the arrow above) are direct measurements (say from LiDAR) and  $\hat{T}$  are obtained from ICP. ( $\vec{X}_{0p}$  below could also refer to [1. described above](#) but for the sake of this derivation, we will keep things simple and refer to it as direct LiDAR measurement).

$$\sum_{i=1}^m \left\| \vec{X}_{ip} - \hat{\mathbf{T}}_{i0} \vec{X}_{0p} \right\|_2^2$$

Over  $n$  points from  $j = 1 \rightarrow n$  (Remember  $m$  is frames)

$$\sum_{i=1}^m \sum_{j=1}^n \left\| \vec{X}_{ij} - \hat{\mathbf{T}}_{i0} \vec{X}_{0j} \right\|_2^2 \quad (1.1)$$

**What are the optimum variables: the 3D points  $\vec{X}_{0j}, j = 1 \rightarrow n$  and the locations of the Mobile Robot  $\hat{\mathbf{T}}_{0i}, i = 1 \rightarrow m$  so that  $\sum_{i=1}^m \sum_{j=1}^n \left\| \vec{X}_{ij} - \hat{\mathbf{T}}_{i0} \vec{X}_{0j} \right\|_2^2$  or**

$$\sum_{i=1}^m \sum_{j=1}^n \left\| \mathbf{r}_{ij} \right\|_2^2$$

**is minimized?**



Note: You might be having valid concerns about the way this loss function is formulated and what we chose as the optimization variables. As briefly mentioned above, there could be multiple ways to model this, even better than what we did above. But for the sake of this lecture, to keep things simple, we will assume the loss function as per (1.1) and optimization variables as per (1.2). The objective is here is to appreciate the Jacobian structure and the update equations, especially when it involves non-Euclidean variables such as  $\mathbf{T}$ .

## Num of variables

Our optimization variables are

$$\mathbf{R}, \mathbf{t}; \mathbf{X}, \mathbf{T}$$

The number of optimization variables are  $\implies 12M + 3N$

Let our initialization be  $R_{iO}, t_{iO}, X_{jO}$ . Recollect the steps in non-linear least squares optimization.

## Update step

We know that an update in non-linear least squares is  $\beta^{n+1} = \beta^n + \delta\beta$  where  $x$  is the optimization variable. Writing the same for the problem at hand:

$$\begin{aligned} X_{0j}(n+1) &= X_{0j}(n) - \delta X_{0j} \\ X_j(n+1) &= X_j(n) - \delta X_j \quad \text{dropping suffix } 0 \end{aligned}$$

## Update step for $\mathbf{T}$

Similar update step can be written for  $\mathbf{T}_0$  as well as follows:

$$\mathbf{T}_{i0}(n+1)_{3 \times 4} = \mathbf{T}_{i0}(n)_{3 \times 4} - \delta \mathbf{T}_{i0} \quad \text{wrong}$$

▼ Why is this **wrong**?

This is because  $\mathbf{T}$  is not in Euclidean space, and addition (and hence such an update) doesn't make sense there. Think about it: Is sum of two Rotation (or Transformation) matrices a Rotation (or Transformation) matrix? The answer is no. Only the "product" operation is preserved for these entities.

Ah, let's rewind a bit. The update of  $\mathbf{T}$  and  $X$  as localization and mapping updates are very different, for the latter is the regular Euclidean update and the other is over a Manifold.

$\mathbf{T}$  is a matrix and not a vector, so the standard update rule cannot be applied as it is.  $\mathbf{R}$  is actually over-parametrized as we only need 6 parameters (3 rotation and 3 translation), so we can instead write our  $\mathbf{T}$  in the form of "rotation about arbitrary axis" vector  $\xi$ :

💡 Recollect: Rotation about arbitrary axis links: [Notion link](#) and [Moodle file](#).

$$\xi = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

where  $\omega = [\omega_1, \omega_2, \omega_3]^T$  represent the axis about which the rotation occurred

The update step is then written via *exp* mapping of *skew-symmetric* matrix form of the vector  $\xi$ :

$$\mathbf{T}_{i0}(n+1)_{3 \times 4} = \mathbf{T}_{i0}(n)_{3 \times 4} \exp([\delta \xi_{i0}]_{\times}) \quad \checkmark \text{ correct} \quad (\text{L1})$$

(L1) is the **exponential** map that maps a vector in the local tangent space of  $\mathbf{T}$ , denoted by  $\xi$  back to  $\mathbf{T}$ .

- Note that  $[abc]_{\times}$  (same as  $[abc]^{\wedge}$ ): Means the skew-symmetric matrix version of a vector  $abc$ . [See this below](#) for more clarity.

$$\xi_{i0}(n+1) = \log([\mathbf{T}_{i0}(n+1)]^{\vee}) \quad (\text{L2})$$

(L2) is the **logarithmic** map that maps the transform matrix  $\mathbf{T}$  to the local tangent vector  $\xi$ .

- You can think of  $^\vee$  as the "inverse" operator of  $^\wedge$  or  $\times$ , i.e.  $[\omega]_\times^\vee = \omega$ . But fortunately, we don't need to evaluate the log mapping analytically, there's a much simpler way as explained below.

Let's first revisit the weird term in (L1).

## Obtaining the matrix $\mathbf{T}$ from the vector $\xi$ through exp map

$$\exp([\delta\xi_{i0}]_\times) = \begin{bmatrix} R(\delta\omega_{i0}) & J(\delta\omega_{i0})\delta v_{i0} \\ 0^T & 1 \end{bmatrix} \quad (\text{L3})$$

The mapping from  $\omega$  to  $R$  is as follows:

$$R(\omega) = \exp([\omega]_\times) = I_{3 \times 3} + \left(\frac{\sin \theta}{\theta}\right) \omega_\times + \left(\frac{1 - \cos \theta}{\theta^2}\right) \omega_\times^2 \quad (2.0)$$

$\theta$  is the magnitude of the rotation  $\omega =$

$[\omega_1, \omega_2, \omega_3]^T$ .  $[\omega]_\times$  is the skew symmetric matrix of  $\omega$ . And  $\theta^2 = \omega^T \omega$ .

▼ *exponential* of a matrix?

You can just look at it as a formula for now.

For those interested to go deeper, here's a short primer on Lie Algebra.

( will be adding in sometime )

- This looks similar to Rodrigues' Formula you learnt during Transformation lecture, isn't it? Indeed it is, refer to the primer if curious.



$$\begin{aligned}
R(\delta\omega_{i0}) &= \exp([\delta\omega_{i0}]_{\times}) \\
&= \mathbf{I}_{3 \times 3} + \frac{\sin |\delta\omega_{i0}|}{|\delta\omega_{i0}|} [\delta\omega_{i0}]_{\times} + \frac{(1 - \cos |\delta\omega_{i0}|)}{|\delta\omega_{i0}|^2} [\delta\omega_{i0}]_{\times}^2 \quad (\text{L4.1})
\end{aligned}$$

where  $\delta\omega_{i0} = [\delta\omega_{i0\ 1}, \delta\omega_{i0\ 2}, \delta\omega_{i0\ 3}]^T$  is the update to the axis angle and  $|\delta\omega_{i0}|$  is the magnitude of  $\delta\omega_{i0}$  which is  $\theta$  or the rotation about  $\delta\omega_{i0}$ .

Another way to write (L4.1) would be to express  $\delta\omega_{i0} = \theta \mathbf{a}$ , its length and direction, denoted as  $\theta$  and  $\mathbf{a}$ .  $\mathbf{a}$  is a unit-length direction vector, i.e.,  $\|\mathbf{a}\| = 1$ .

$$\begin{aligned}
R(\delta\omega_{i0}) &= \exp([\delta\omega_{i0}]_{\times}) = \exp(\theta \mathbf{a}^{\wedge}) \\
&= (1 - \cos \theta) \mathbf{a}^{\wedge} \mathbf{a}^{\wedge} + \mathbf{I} + \sin \theta \mathbf{a}^{\wedge} \quad (\text{L4.2})
\end{aligned}$$

$$J(\delta\omega_{i0}) = \frac{\sin \theta}{\theta} \mathbf{I} + \left(1 - \frac{\sin \theta}{\theta}\right) \mathbf{a} \mathbf{a}^T + \frac{1 - \cos \theta}{\theta} \mathbf{a}^{\wedge} \quad (\text{L4.3})$$

## Obtaining the vector $\xi$ from the matrix $\mathbf{T}$

$$\xi = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

where  $\omega = [\omega_1, \omega_2, \omega_3]^T$  represent the axis about which the rotation occurred

### Finding $\omega$ 's:

*log map isn't necessary here, we can do it in simpler way. Let's drop additional notations in this section for simplicity (like  $\delta$ 's or sub/superscripts).*

We can first simply take the top left corner of our  $\mathbf{T}$  to get  $R$ . Now that we have  $R$ ,

$$\omega = \begin{bmatrix} \omega_1 & \omega_2 & \omega_3 \end{bmatrix}^T = \log(R) \quad (\text{L5})$$

But I promised *log map*, isn't necessary —

💡 Remember when we studied Rodrigues' Formula? **Without** evaluating the *log map*, we can actually find the rotation vector  $\omega$  given  $R$  rotation matrix. [Revisit this link](#).

---

**Now we got the first 3 parameters of  $\xi$ . What about the  $v$ 's?**

The upper right corner gives:

$$t = Jv$$

And hence,

$$v = \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix}^T = J^{-1}t \quad (\text{L6})$$

**Remember that we already found out  $\omega$  so  $J$  can be found.**

For  $\mathbf{T}_{i0}$ ,

$$\omega_{i0} = \begin{bmatrix} \omega_{i0\ 1} & \omega_{i0\ 2} & \omega_{i0\ 3} \end{bmatrix}^T = \log(R_{i0})$$

$$v_{i0} = \begin{bmatrix} v_{i0\ 1} & v_{i0\ 2} & v_{i0\ 3} \end{bmatrix}^T = R_{i0}^{-1}t_{i0}$$

As we will see below, the Jacobian is evaluated in the next iteration w.r.t  $\xi_{i0}(n+1)$ ,  $X_{0j}(n+1)$  where  $i = 1 \rightarrow m, j = 1 \rightarrow n$ .

---

## Moving forward with calculating Jacobian

💡 We know that we can find that  $\delta X_j$  or  $\delta \xi_i$  by calculating the corresponding Jacobian. Below, whenever we refer to  $\delta_{ij}$ , think of it as the overall update vector got by stacking those 2 vectors  $\delta_{ij} = [\delta X_j \mid \delta \xi_i]$ .

💡 Recollect that the update for gradient descent is given by  $\delta \mathbf{x} = -\alpha \mathbf{J}_{\mathbf{r}}^T \mathbf{r}(\mathbf{x})$ . Similarly for Gauss-Newton or LM, if we know the Jacobian (and residual) at the current estimate, we can find our update value. Stacking the two Jacobians together, we get a  $3 \times 15$  matrix as follows:

$$\mathbf{J}_{ij} \text{ (3} \times \text{15)} = \begin{bmatrix} \frac{\partial \mathbf{r}_{ij}}{\partial \mathbf{T}_i \text{ (3} \times \text{12)}} & \frac{\partial \mathbf{r}_{ij}}{\partial \mathbf{X}_j \text{ (3} \times \text{3)}} \end{bmatrix} \quad \otimes \text{ not completely right (1.3)}$$

Let's recollect how we ended up with  $\frac{\partial \mathbf{r}_{ij}}{\partial \mathbf{X}_j}$  as  $3 \times 3$  matrix:  $\mathbf{r}$  is a 3 dimensional **vector** and so is  $\mathbf{X}$ , therefore our Jacobian would be  $3 \times 3$ .

Now coming to  $\frac{\partial \mathbf{r}_{ij}}{\partial \mathbf{T}_i}$ , something looks fishy, isn't it?

$\mathbf{T}$  is a **matrix**  $\in \text{SE}(3)$ , if we flatten it out, we'll end up with a 12-dimensional **vector**  $\in \mathbb{R}^{12}$  and thus we'll end up with  $3 \times 12$  Jacobian matrix. However, by flattening, we'd be losing the useful properties of a transform matrix (For example, the fact that columns of the rotation matrix are orthogonal to one another). So a natural question arises:

| Is there a better way to do this?

$\mathbf{R}$  is actually over-parametrized as we only need 6 parameters (3 rotation and 3 translation), so we can instead write our  $\mathbf{T}$  in the form of "rotation about arbitrary axis" vector as a function of  $\xi$ :

💡 Recollect: Rotation about arbitrary axis links: [Moodle file](#) and [Notion link](#).

»

and hence our residual would be:

$$\mathbf{r}_{ij}(\mathbf{T}_i(\xi_i), \mathbf{X}_j) \quad (1.4.1)$$

Therefore, we instead take derivative of residual w.r.t  $\xi$ :

$$\mathbf{J}_{ij} \text{ (3} \times \text{9)} = \begin{bmatrix} \left( \frac{\partial \mathbf{r}_{ij}}{\partial \mathbf{T}_i} \frac{\partial \mathbf{T}_i}{\partial \xi_i} \right)_{(3 \times 6)} & \frac{\partial \mathbf{r}_{ij}}{\partial \mathbf{X}_j \text{ (3} \times \text{3)}} \end{bmatrix} \quad \checkmark \text{ correct (1.4.2)}$$

Read [more description below](#) for more clarity.

$$\mathbf{J}_{ij} \text{ (3} \times \text{9)} = \begin{bmatrix} \left( I_3 \text{ (3} \times \text{3)} - [\mathbf{T} \oplus \mathbf{X}_j]_{\times} \text{ (3} \times \text{3)} \right)_{(3 \times 6)} & \frac{\partial \mathbf{f}_{ij}}{\partial \mathbf{X}_j \text{ (3} \times \text{3)}} \end{bmatrix} \quad (1.5.1)$$

This derivation ( $I - [T \oplus X]$ ) can be seen in the paper "Least Squares Optimization: from Theory to Practice Giorgio Grisetti"

You don't need to worry too much about how we went from (1.4.2) to 1.5.1. But let's address those weird symbols  $\oplus$  and  $\times$  in 1.5.1 though.

▼ What is  $\oplus$ ? (For Dummies: Click on Toggle )

It is a pose composition whose can simply be written as a *matrix*  $\times$  *vector* as follows:

$$T \oplus X_j = TX_j$$

The product  $TX_j$  is now a vector. For the scope of this course, this much understanding is enough.

▼ What is  $\times$  in  $[TX_j]_{\times}$  then?

$[TX_j]_{\times}$  is a skew symmetric matrix version of  $[TX_j]$

▼ 💡 Revisiting *skew-symmetric matrix & cross product*:

If  $P = [x, y, z]^T$  is a vector, its skew symmetric form is given by:

$$[P]_{\times} = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}$$

• *Cross product as skew-symmetric matrix*:

$$\begin{aligned} \mathbf{a} \times \mathbf{b} &= \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} \\ &= (a_2b_3 - a_3b_2)\mathbf{i} + (a_3b_1 - a_1b_3)\mathbf{j} + (a_1b_2 - a_2b_1)\mathbf{k} \end{aligned}$$

- - - -

$$\mathbf{a} \times \mathbf{b} = [\mathbf{a}]_{\times} \mathbf{b} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$\mathbf{a} \times \mathbf{b} = [\mathbf{b}]_{\times}^T \mathbf{a} = \begin{bmatrix} 0 & b_3 & -b_2 \\ -b_3 & 0 & b_1 \\ b_2 & -b_1 & 0 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

So,

$$\mathbf{J}_{ij} \text{ (3} \times \text{9)} = \begin{bmatrix} \left( I_3 \text{ (3} \times \text{3)} - [\mathbf{TX}_j]_{\times} \text{ (3} \times \text{3)} \right) \text{ (3} \times \text{6)} & \frac{\partial \mathbf{f}_{ij}}{\partial \mathbf{X}_j} \text{ (3} \times \text{3)} \end{bmatrix} \quad (1.5.2)$$

And our update would be  $\delta_{ij} \text{ (9} \times \text{1)}$ .

$\mathbf{J}_{ij}$  can be split as:

$$\mathbf{J}_{ij} = \begin{bmatrix} \underbrace{\mathbf{J}_{ij \text{ L}} \text{ (3} \times \text{6)}}_{\text{Localization Jacobian}} \mid \underbrace{\mathbf{J}_{ij \text{ M}} \text{ (3} \times \text{3)}}_{\text{Mapping Jacobian}} \end{bmatrix}$$

- **Localization Jacobian:** Associated with the pose derivatives
  - $\mathbf{J}_{11 \text{ L}}$  is the Jacobian obtained by taking the derivative with respect to  $\hat{\mathbf{T}}_{10}$  (**actually**  $\xi_{10}$ ) of the term  $\left[ \vec{X}_{11} - \hat{\mathbf{T}}_{10} \vec{X}_{01} \right]$ .
  - $\mathbf{J}_{12 \text{ L}}$  w.r.t  $\hat{\mathbf{T}}_{10}$  (**actually**  $\xi_{10}$ ) of  $\left[ \vec{X}_{12} - \hat{\mathbf{T}}_{10} \vec{X}_{02} \right]$
  - ...
  - $\mathbf{J}_{mn \text{ L}}$  w.r.t  $\hat{\mathbf{T}}_{m0}$  (**actually**  $\xi_{m0}$ ) of  $\left[ \vec{X}_{mn} - \hat{\mathbf{T}}_{m0} \vec{X}_{0n} \right]$ 
    - $\xi_{m0}$  is the **local tangent vector** of  $\hat{\mathbf{T}}_{m0}$  obtained from "**Log map**".
- **Mapping Jacobian:** Associate with the map (point cloud) derivatives

- $\mathbf{J}_{11M}$  is the Jacobian obtained by taking the derivative with respect to  $\vec{X}_{01}$  of the term  $[\vec{X}_{11} - \hat{\mathbf{T}}_{10}\vec{X}_{01}]$ .
- $\mathbf{J}_{12M}$  w.r.t  $\vec{X}_{02}$  of  $[\vec{X}_{12} - \hat{\mathbf{T}}_{10}\vec{X}_{02}]$
- ...
- $\mathbf{J}_{mnM}$  w.r.t  $\vec{X}_{0n}$  of  $[\vec{X}_{mn} - \hat{\mathbf{T}}_{m0}\vec{X}_{0n}]$

Then,



**Our overall Jacobian** for all  $m$  poses and  $n$  points would be (Remember: *pose*  $i = 1 \rightarrow m$ , *points*  $j = 1 \rightarrow n$ )

$$\mathbf{J} = \begin{bmatrix} \overbrace{\mathbf{J}_{11L} \quad 0 \quad \dots \quad 0}^{\text{'m' times}} & \overbrace{\mathbf{J}_{11M} \quad 0 \quad \dots \quad 0}^{\text{'n' times}} \\ \mathbf{J}_{12L} \quad 0 \quad \dots \quad 0 & 0 \quad \mathbf{J}_{12M} \quad \dots \quad 0 \\ \vdots & \vdots \\ \mathbf{J}_{1nL} \quad 0 \quad \dots \quad 0 & 0 \quad \dots \quad \mathbf{J}_{1nM} \\ 0 \quad \mathbf{J}_{21L} \quad \dots \quad 0 & \mathbf{J}_{21M} \quad 0 \quad \dots \quad 0 \\ \vdots & \vdots \\ 0 \quad \mathbf{J}_{2nL} \quad \dots \quad 0 & 0 \quad \dots \quad \mathbf{J}_{2nM} \\ \vdots & \vdots \\ 0 \quad \dots \quad \mathbf{J}_{m1L} & \mathbf{J}_{m1M} \quad \dots \quad 0 \\ \vdots & \vdots \\ 0 \quad \dots \quad \mathbf{J}_{mnL} & 0 \quad \dots \quad \mathbf{J}_{mnM} \end{bmatrix} \begin{bmatrix} \delta \xi_1 \\ \delta \xi_2 \\ \vdots \\ \delta \xi_m \\ \delta X_{01} \\ \delta X_{02} \\ \vdots \\ \delta X_{0n} \end{bmatrix}$$

**LOCALIZATION JACOBIAN**
**MAPPING JACOBIAN**

Only the left matrix is  $\mathbf{J}$ . The  $\delta$  vector is written on the right. It's **NOT** multiplication: Just written together for convenience. (Although unintentionally written in the figure, notice how you can actually multiply them, i.e. I mean to talk about their matching dimensionality — No. of columns in  $\mathbf{J} = \dim(\delta)$ . Why is this true? Convince yourself. 🤖)

## Summarizing the story so far...

Recollect our update  $\delta = -\alpha \mathbf{J}^\top \mathbf{r}$ .

- For every one of the  $m$  robot locations, there are  $n$  points giving rise to  $3n$  ICP equations. Note each point gives 3 ICP equations.
- For  $m$  such robot locations, we have  **$3nm$  equations**.
- Each of the  $m$  poses has 6 parameters in the tangent vector  $\xi_i$  and each point has 3 components:  **$6m + 3n$  variables**

Summarizing the dimensions of all 3 entities in  $\delta = -\alpha \mathbf{J}^\top \mathbf{r}$ :

$$\delta_{(6m+3n)} \mid \mathbf{J}_{(3mn, 6m+3n)} \mid \mathbf{r}_{3mn}$$

## Revisiting our residual error function & update

$$\sum_{i=1}^m \sum_{j=1}^n \left\| \vec{X}_{ij} - \hat{\mathbf{T}}_{i0} \vec{X}_{0j} \right\|_2^2$$

Using Gauss Newton, the update can be written as:

$$\underbrace{\begin{bmatrix} \delta \xi_i \\ \delta x_j \end{bmatrix}}_{(6m+3n, 1)} = \underbrace{[\mathbf{J}^\top \mathbf{J}]^{-1}}_{(6m+3n, 3mn) \times (3mn, 6m+3n)} \underbrace{\mathbf{J}^\top}_{(6m+3n, 3mn)} \underbrace{\left[ \vec{X}_{ij} - \hat{R}_{i0} \vec{X}_{0j} + \hat{t}_{i0} \right]}_{(3mn, 1)}$$

- Not necessary to code these from scratch during practical applications.
- Solvers like ceres, G2O, GTSAM will do these for you.
- **But you need to appreciate the cost function, the Jacobian structure and the general notion of why we resort to Manifold optimization.**