ICP-SLAM as LS Optimization problem

Comments	
Dates Taught	@September 15, 2020 → September 22, 2020
Lecture No.	L10 L11 L12
Links of Videos	L10, L11 (Sep 18), L12
Module	SLAM: Smoothing
Related to All Questions (Property)	

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

> Note-1: Please read this note first.

> This page is complete and this level of understanding is sufficient for the scope of this course. **However, the way it is introduced might be a bit confusing. Therefore, it is highly suggested to go through the basics first (LINK TO BE ADDED) and understand the general formulation of any common optimization function in robotics/vision.**
>
> For interested readers, there are certain Lie Group/Lie Algebra concepts which may be given as formulae in this page. You can refer to "A micro Lie theory for state estimation in robotics" to dive into more detail.

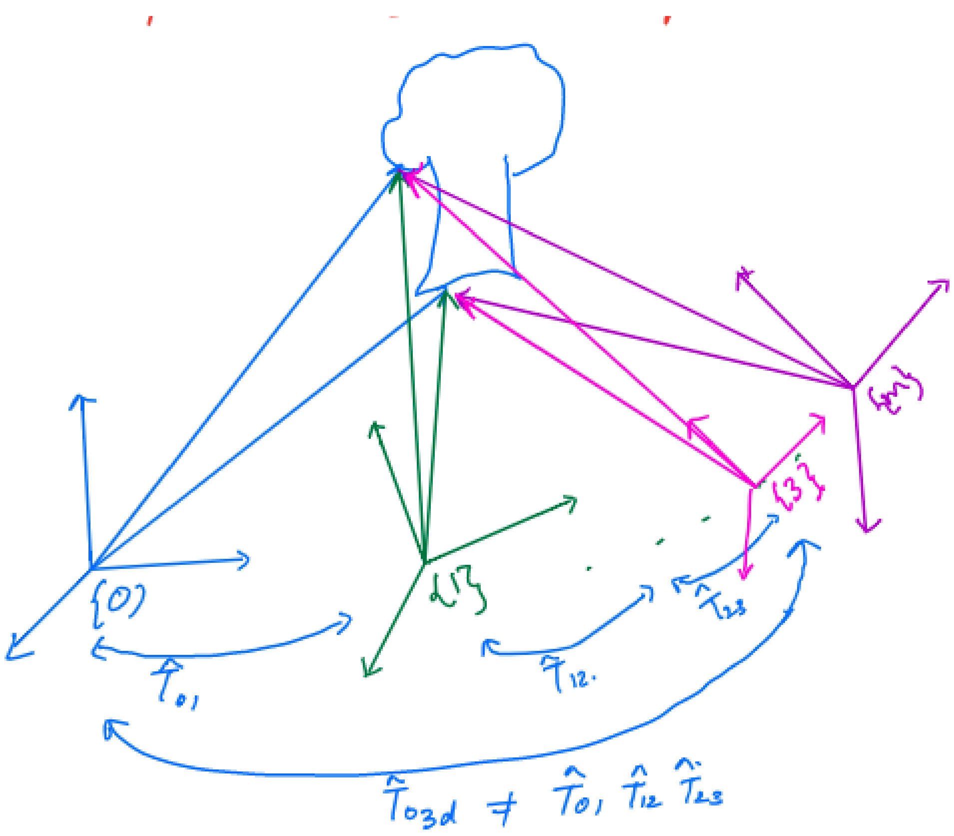
Prof's notes on ICP-SLAM

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 $\widehat{X}_{ij}$ 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 ξ through $\exp$ map
Obtaining the vector ξ 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 $\{0\}$ 🖼️ —— $\mathrm{T}_0 = [\mathbf{I}|\mathbf{0}]$ is the origin.
- Blue tree 🌴—— $\mathrm{X}_{0j} = \begin{bmatrix} x_{0j} & y_{0j} & z_{0j} \end{bmatrix}^{\mathrm{T}}$: where $j = \{1, \ldots, n\}$ are the coordinates of n points in frame $\{0\}$ obtained from depth, azimuth and elevation measurements.
  - Since the depth for every pont 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. $\widehat{\mathrm{T}}_{01}, \widehat{\mathrm{T}}_{12}, \ldots, \widehat{\mathrm{T}}_{(m-1,\, m)}$ 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 $j = \{1, \ldots, n\}$ be represented in frames $i = \{1, \ldots, m\}$ as $\mathrm{X}_{ij} = \begin{bmatrix} x_{ij} & y_{ij} & z_{ij} \end{bmatrix}^{\mathrm{T}}$ as mn observations. $\widehat{\mathrm{X}}_{ij}$ **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 $\widehat{\mathrm{T}}_{0i}$ **estimated from ICP** as $\widehat{\mathrm{T}}_{0i} = \widehat{\mathrm{T}}_{01}\widehat{\mathrm{T}}_{12}\ldots\widehat{\mathrm{T}}_{(i-1,\, i)}$ we predict what is X_{0j} in frame i as

$$\widehat{\mathrm{X}}_{ij} = \widehat{\mathrm{T}}_{i0}\mathrm{X}_{0j} \tag{0.1}$$

Why is this $\widehat{\mathrm{X}}_{ij}$ 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 $\widehat{\mathrm{X}}^i_{0j} = \widehat{\mathrm{T}}_{0i}\mathrm{X}_{ij}$, I get m sets of n points in frame $\{0\}$ that I average as

$$\widehat{\mathrm{X}}_{0j} = \sum_{i=1}^{m} \frac{\widehat{\mathrm{X}}^i_{0j}}{m} \tag{0.2}$$

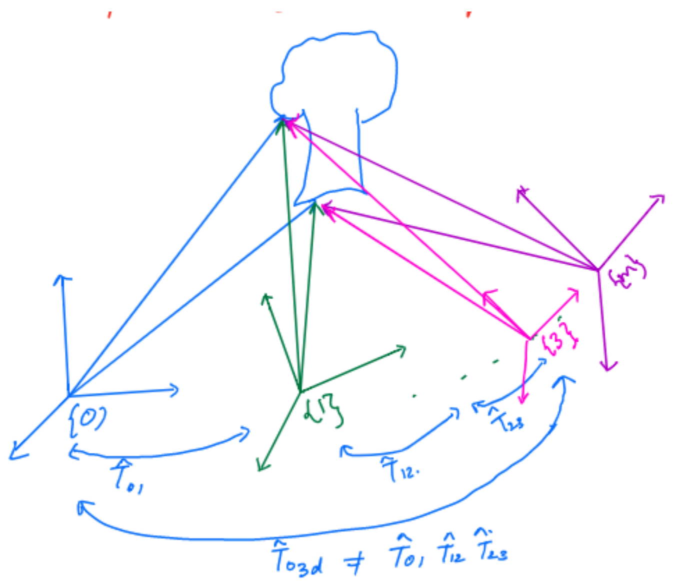
   or such aggregation over the m frames.

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

$$\widehat{\mathrm{X}}_{ij\mathbf{d}} = \widehat{\mathrm{T}}_{i0\mathbf{d}}\mathrm{X}_{0j} \tag{0.3}$$

   should be the same as $\widehat{\mathrm{X}}_{ij}$ 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 $\widehat{T}$ 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).

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

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

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

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

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

is minimized?

> 💡 Note: You might be having some confusion about the way this loss function is formulated and what we chose as the optimization variables. I**t is highly suggested to go through the basics first (LINK TO BE ADDED) and understand the general formulation of any common optimization function in robotics/vision.**

Num of variables

Our optimization variables are

$$", "; ", ""$$

The number of optimization variables are $\Longrightarrow$ $\boxed{12\text{M} + 3\text{N}}$

Let our initialization be R_{iO}, t_{iO}, X_{j_O}. 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 β is the optimization variable. Writing the same for the problem at hand:

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

Update step for $\mathbf{T}$

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

$$\cancel{\mathbf{T}_{i0}(n+1)_{3\times 4} = \mathbf{T}_{i0}(n)_{3\times 4} - \delta\mathbf{T}_{i0}} \qquad \otimes\ \textbf{wrong}$$

▼ 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. Simply put, sum of two Transformation matrices is not a Transformation matrix.

$\mathbf{T}$ 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 ξ :

💡 Recollect: Rotation about arbitrary axis links: Notion link and Moodle file.

$$\xi = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix}$$
$$\text{where } \omega = [\omega_1, \omega_2, \omega_3]^T \text{ represent the axis}$$
$$\text{about which the rotation occurred}$$

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

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

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

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

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

- You can think of $^\vee$ as the "inverse" operator of $^\wedge$ or $\times$, i.e. $[\omega]^\vee_\times = \omega$. 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 ξ through $\exp$ map

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

The mapping from ω to R is as follows:

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

θ is the magnitude of the rotation $\omega = [\omega_1, \omega_2, \omega_3]^T$. $[\omega]_\times$ is the skew symmetric matrix of ω. 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.

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

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 θ 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 θ and $\mathbf{a}$. $\mathbf{a}$ is a unit-length direction vector, i.e., $||\mathbf{a}|| = 1$.

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

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

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

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

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

> ### *Finding ω'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 δ'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,

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

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 ω given R rotation matrix. Revisit this link.

> ### *Now we got the first 3 parameters of ξ. What about the v's?*

The upper right corner gives:

$$t = Jv$$

And hence,

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

Remember that we already found out ω 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 δX_j or $\delta \xi_i$ by calculating the corresponding Jacobian. Below, whenever we refer to δ_{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 $\delta \mathbf{x} = -\alpha \mathbf{J_r}^\top \mathbf{r}(\mathbf{x})$. 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 3×15 matrix as follows:

$$\mathbf{J}_{ij\ (3\times15)} = \begin{bmatrix} \dfrac{\partial \mathbf{r_{ij}}}{\partial \mathbf{T}_i}_{\ (3\times12)} & \dfrac{\partial \mathbf{r_{ij}}}{\partial X_j}_{\ (3\times3)} \end{bmatrix} \quad \text{not completely right} \tag{1.3}$$

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

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

$\mathbf{T}$ is a **matrix** $\in \mathrm{SE}(3)$, 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 3×12 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 ξ:

💡Recollect: Rotation about arbitrary axis links: Moodle file and Notion link.

and hence our residual would be:

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

Therefore, we instead take derivative of residual w.r.t ξ:

$$\mathbf{J}_{ij\ (3\times 9)} = \left[\left(\frac{\partial \mathbf{r_{ij}}}{\partial \mathbf{T}_i} \frac{\partial \mathbf{T}_i}{\partial \xi_\mathbf{i}} \right)_{(3\times 6)} \quad \frac{\partial \mathbf{r_{ij}}}{\partial \mathbf{X}_j}_{\ (3\times 3)} \right] \qquad \checkmark \textbf{correct} \tag{1.4.2}$$

Read more description below for more clarity.

$$\mathbf{J}_{ij\ (3\times 9)} = \left[\left(I_{3\ (3\times 3)} \ \mid\ -[\mathrm{T} \oplus \mathrm{X_j}]_{\times\ (3\times 3)} \right)_{(3\times 6)} \quad \frac{\partial \mathbf{f_{ij}}}{\partial \mathbf{X}_j}_{\ (3\times 3)} \right] \tag{1.5.1}$$

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:

$$\mathrm{T} \oplus \mathrm{X_j} = \mathrm{TX_j}$$

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

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

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

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

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

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

- *Cross product as skew-symmetric matrix:*

$$\begin{aligned} \mathbf{a} \times \mathbf{b} &= \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} \\ &= (a_2 b_3 - a_3 b_2)\,\mathrm{i} + (a_3 b_1 - a_1 b_3)\,\mathrm{j} + (a_1 b_2 - a_2 b_1)\,\mathrm{k} \end{aligned}$$

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

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

So,

$$\mathbf{J}_{ij\ (3\times 9)} = \left[\left(I_{3\ (3\times 3)} \ \mid\ -[\mathrm{TX_j}]_{\times\ (3\times 3)} \right)_{(3\times 6)} \quad \frac{\partial \mathbf{f_{ij}}}{\partial \mathbf{X}_j}_{\ (3\times 3)} \right] \tag{1.5.2}$$

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

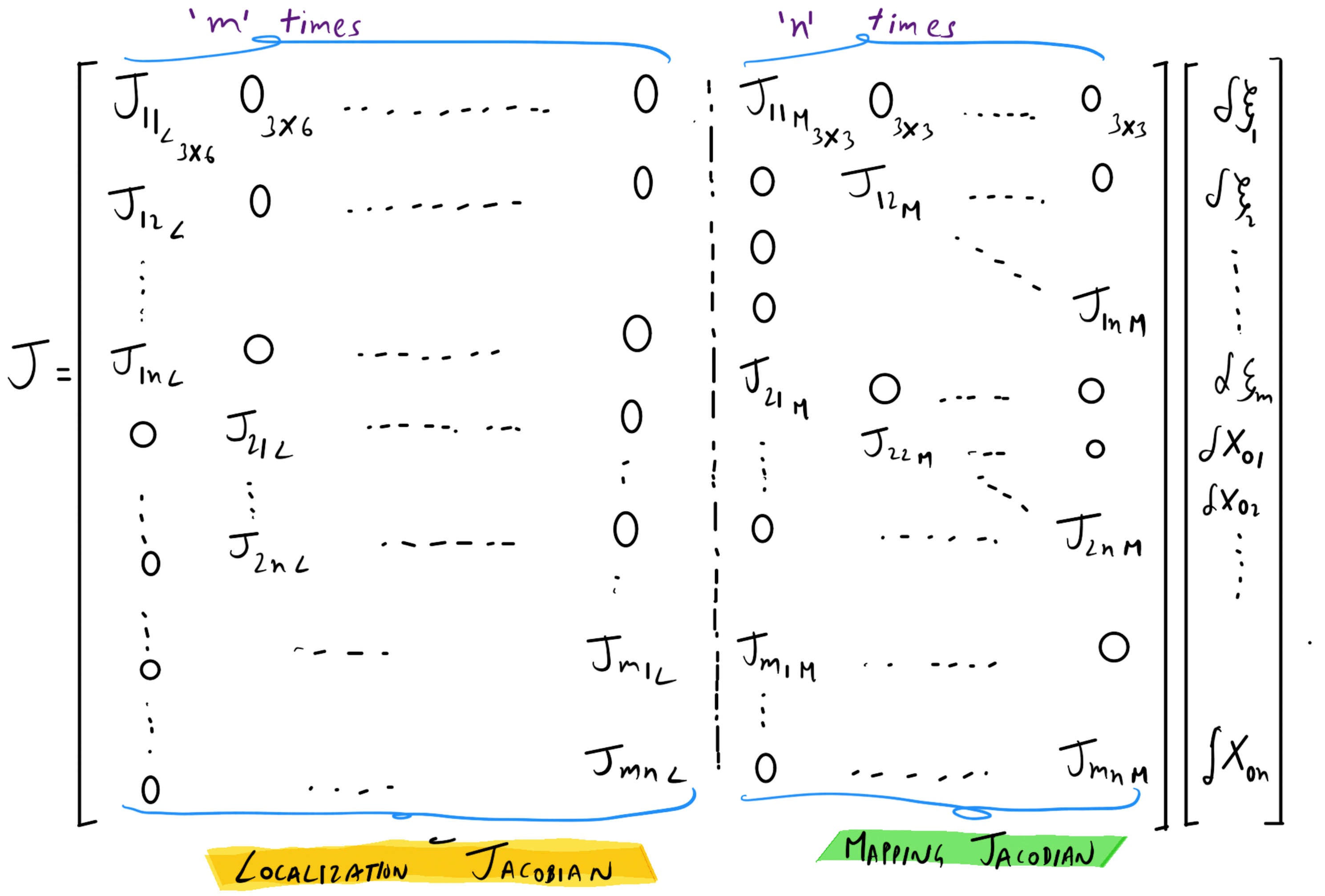
$\mathbf{J}_{ij}$ can be split as:

$$\mathbf{J}_{ij} = \left[\underbrace{\mathbf{J_{ij\ L}}_{(3\times 6)}}_{Localization\ Jacobian} \ \mid\ \underbrace{\mathbf{J_{ij\ M}}_{(3\times 3)}}_{Mapping\ Jacobian} \right]$$

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

Then,

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



Only the left matrix is $\mathbf{J}$. The δ vector is written on the right. It's **NOT** multiplication: Just written together for convenience (You can compare and see the parameters corresponding to each Jacobian though). (Notice how you can actually multiply them, i.e. see their matching dimensionality — $\text{No. of columns in } \mathbf{J} = dim(\delta)$. 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 ξ_i and each point has 3 components: $\text{6m} + \text{3n variables}$

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

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

Using Gauss Newton, the update can be written as:

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

- 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.**
