6.801/6.866: Machine Vision, Lecture 19

Professor Berthold Horn, Ryan Sander, Tadayuki Yoshitake
MIT Department of Electrical Engineering and Computer Science
Fall 2020

These lecture summaries are designed to be a review of the lecture. Though I do my best to include all main topics from the lecture, the lectures will have more elaborated explanations than these notes. Therefore, if you're looking for the most rigorous review and treatment of these topics, we encourage you to rewatch the lecture videos. With that said, we hope these summaries are beneficial for your learning. If you have any feedback for these lecture summaries, please submit it **here**.

1 Lecture 19: Absolute Orientation in Closed Form, Outliers and Robustness, RANSAC

This lecture will continue our discussion of photogrammetry topics - specifically, covering more details with the problem of **absolute orientation**. We will also look at the effects of outliers on the robustness of closed-form absolute orientation, and how algorithms such as RANSAC can be leveraged as part of an absolute orientation, or, more generally, photogrammetry pipeline to improve the robustness of these systems.

1.1 Review: Absolute Orientation

Recall our four main problems of photogrammetry:

- **Absolute Orientation** $3D \longleftrightarrow 3D$
- **Relative Orientation** $2D \longleftrightarrow 2D$
- **Exterior Orientation** $2D \longleftrightarrow 3D$
- **Intrinsic Orientation** $3D \longleftrightarrow 2D$

In the last lecture, we saw that when solving **absolute orientation** problems, we are mostly interested in finding **transformations** (translation + rotation) between two coordinate systems, where these coordinate systems can correspond to objects or sensors moving in time (recall this is where we saw duality between objects and sensors).

Last time, we saw that one way we can find an optimal **transformation** between two coordinate systems in 3D is to decompose the optimal transformation into an optimal **translation** and an optimal **rotation**. We saw that we could solve for optimal translation in terms of rotation, and that we can mitigate the constraint issues with solving for an orthonormal rotation matrix by using **quaternions** to carry out rotation operations.

1.1.1 Rotation Operations

Relevant to our discussion of quaternions is identifying the critical operations that we will use for them (and for orthonormal rotation matrices). Most notably, these are:

1. **Composition of rotations**: $\overset{o}{p}\overset{o}{q} = (p, \mathbf{q})(q, \mathbf{q}) = (pq - \mathbf{q} \cdot \mathbf{q}, p\mathbf{q} + q\mathbf{q} + \mathbf{q} \times \mathbf{q})$
2. **Rotating vectors**: $\overset{o}{r}' = \overset{o}{q}\overset{o}{r}\overset{o}{q}^* = (q^2 - \mathbf{q} \cdot \mathbf{q})\mathbf{r} + 2(\mathbf{q} \cdot \mathbf{r})\mathbf{q} + 2q(\mathbf{q} \times \mathbf{r})$

Recall from the previous lecture that operation **(1) was faster** than using orthonormal rotation matrices, and operation **(2) was slower**.

1.1.2 Quaternion Representations: Axis-Angle Representation and Orthonormal Rotation Matrices

From our previous discussion, we saw that another way we can represent quaternions is through the axis-angle notation (known as the Rodrigues formula):

$$\mathbf{r}' = (\cos\theta)\mathbf{r} + (1-\cos\theta)(\hat{\boldsymbol{\omega}}\cdot\mathbf{r})\hat{\boldsymbol{\omega}} + \sin\theta(\hat{\boldsymbol{\omega}}\times\mathbf{r})$$

Combining these equations from above, we have the following axis-angle representation:

$$\overset{o}{q} \Longleftrightarrow \hat{\boldsymbol{\omega}},\theta,\;\; q = \cos\left(\frac{\theta}{2}\right),\; \mathbf{q} = \hat{\boldsymbol{\omega}}\sin\left(\frac{\theta}{2}\right) \implies \overset{o}{q} = \left(\cos\left(\frac{\theta}{2}\right), \hat{\boldsymbol{\omega}}\sin\left(\frac{\theta}{2}\right)\right)$$

We also saw that we can convert these quaternions to orthonormal rotation matrices. Recall that we can write our vector rotation operation as:

$$\overset{o}{q}\overset{o}{r}\overset{o}{q}{}^{*} = (\bar{Q}^TQ)\overset{o}{r}, \text{ where}$$

$$\bar{Q}^TQ = \begin{bmatrix} \overset{o}{q}\cdot\overset{o}{q} & 0 & 0 & 0 \\ 0 & q_0^2+q_x^2-q_y^2-q_z^2 & 2(q_xq_y-q_0q_z) & 2(q_xq_z+q_0q_y) \\ 0 & 2(q_yq_x+q_0q_z) & q_0^2-q_x^2+q_y^2-q_z^2 & 2(q_yq_z-q_0q_x) \\ 0 & 2(q_zq_x-q_0q_y) & 2(q_zq_y+q_0q_z) & q_0^2-q_x^2-q_y^2+q_z^2 \end{bmatrix}$$

The matrix $\bar{Q}^TQ$ has **skew-symmetric** components and **symmetric components**. This is useful for conversions. Given a **quaternion**, we can compute orthonormal rotations more easily. For instance, if we want an **axis** and **angle** representation, we can look at the lower right 3×3 submatrix, specifically its trace:

$$\text{Let } R = [\bar{Q}^TQ]_{3\times3,\text{lower}}, \text{ then :}$$

$$\begin{aligned} \text{tr}(R) &= 3q_0^2 - (q_x^2+q_y^2+q_z^2) \\ &= 3\cos^2\left(\frac{\theta}{2}\right) - (\sin^2\left(\frac{\theta}{2}\right) \textbf{ (Substituting our axis-angle representation)} \\ &- \cos^2\left(\frac{\theta}{2}\right) + \sin^2\left(\frac{\theta}{2}\right) - 1 \textbf{ (Subtracting Zero)} \\ &\to 2\cos^2\left(\frac{\theta}{2}\right) - \sin^2\left(\frac{\theta}{2}\right) - 1 \\ &= 2\cos\theta - 1 \\ &\implies \cos\theta = \frac{1}{2}(\text{tr}(R)-1) \end{aligned}$$

While this is one way to get the angle (we can solve for θ through arccos of the expression above), it is not the best way to do so: we will encounter problems near $\theta \approx 0, \pi$. Instead, we can use the **off-diagonal** elements, which depend on $\sin\left(\frac{\theta}{2}\right)$ instead. Note that this works because at angles θ where $\cos\left(\frac{\theta}{2}\right)$ is "bad" (is extremely sensitive), $\sin\left(\frac{\theta}{2}\right)$ is "good" (not as sensitive), and vice versa.

1.2 Quaternion Transformations/Conversions

Next, let us focus on how we can convert between quaternions and orthonormal rotation matrices. Given a 3×3 orthonormal rotation matrix r, we can compute sums and obtain the following system of equations:

$$\begin{aligned} 1 + r_{11} + r_{22} + r_{33} &= 4q_0^2 \\ 1 + r_{11} - r_{22} - r_{33} &= 4q_x^2 \\ 1 - r_{11} + r_{22} - r_{33} &= 4q_y^2 \\ 1 - r_{11} - r_{22} + r_{33} &= 4q_z^2 \end{aligned}$$

This equation can be solved by taking square roots, but due to the number of solutions (8 by Bezout's theorem, allowing for the flipped signs of quaternions, we should not use this set of equations alone to find the solution).

Instead, we can compute these equations, evaluate them, take the largest for numerical accuracy, arbitrarily select to use the positive version (since there is sign ambiguity with the signs of the quaternions), and solve for this. We will call this selected righthand side q_i.

For off-digaonals, which have **symmetric** and **non-symmetric** components, we derive the following equations:

$$
\begin{aligned}
r_{32} - r_{23} &= 4q_0q_x \\
r_{13} - r_{31} &= 4q_0q_y \\
r_{21} - r_{12} &= 4q_0q_x \\
r_{21} + r_{12} &= 4q_xq_y \\
r_{32} + r_{23} &= 4q_yq_z \\
r_{13} + r_{31} &= 4q_zq_z
\end{aligned}
$$

Adding/subtracting off-diagonals give us 6 relations, of which we only need 3 (since we have 1 relation from the diagonals). For instance, if we have $q_i = q_y$, then we pick off-diagonal relations involving q_y, and we solve the four equations given by:

$$
\begin{aligned}
1 - r_{11} + r_{22} - r_{33} &= 4q_y^2 \\
r_{13} - r_{31} &= 4q_0q_y \\
r_{32} + r_{23} &= 4q_yq_z \\
r_{13} + r_{31} &= 4q_zq_z
\end{aligned}
$$

This system of four equations gives us a direct way of going from quaternions to an orthonormal rotation matrix. Note that this could be 9 numbers that could be noisy, and we want to make sure we have best fits.

1.3 Transformations: Incorporating Scale

Thus far, for our problem of absolute orientation, we have considered **transformations** between two coordinate systems of being composed of **translation** and **rotation**. This is often sufficient, but in some applications and domains, such as satellite imaging fr topographic reconstruction, we may be able to better describe these transformations taking account not only **translation** and **rotation**, but also **scaling**.

Taking scaling into account, we can write the relationship between two point clouds corresponding to two different coordinate systems as:

$$\mathbf{r}'_r = sR(\mathbf{r}'_l)$$

Where rotation is again given by $R \in \mathbf{SO}(3)$, and the scaling factor is given by $s \in \mathbb{R}^+$ (where $\mathbb{R}^+ \triangleq \{x : x \in \mathbb{R}\ , x > 0\}$). Recall that $\mathbf{r}'_r$ and $\mathbf{r}'_l$ are the centroid-subtracted variants of the point clouds in both frames of reference.

1.3.1 Solving for Scaling Using Least Squares: Asymmetric Case

As we did before, we can write this as a least-squares problem over the scaling parameter s:

$$\min_s \sum_{i=1}^n ||\mathbf{r}'_{r,i} - sR(\mathbf{r}'_{l,i})||_2^2$$

As we did for translation and rotation, we can solve for an optimal scaling parameter:

$$
\begin{aligned}
s^* &= \arg\min_s \sum_{i=1}^n ||\mathbf{r}'_{r,i} - sR(\mathbf{r}'_{l,i})||_2^2 \\
&= \arg\min_s \sum_{i=1}^n \left(||\mathbf{r}'_{r,i}||_2^2\right) - 2s\sum_{i=1}^n \left(\mathbf{r}'_{r,i}R(\mathbf{r}'_{l,i})\right) + s^2\sum_{i=1}^n ||R(\mathbf{r}'_{l,i})||_2^2 \\
&= \arg\min_s \sum_{i=1}^n \left(||\mathbf{r}'_{r,i}||_2^2\right) - 2s\sum_{i=1}^n \left(\mathbf{r}'_{r,i}R(\mathbf{r}'_{l,i})\right) + s^2\sum_{i=1}^n ||\mathbf{r}'_{l,i}||_2^2 \quad \textbf{(Rotation preserves vector lengths)}
\end{aligned}
$$

Next, let us define the following terms:

1. $s_r \triangleq \sum_{i=1}^n \left(||\mathbf{r}'_{r,i}||_2^2\right)$
2. $D \triangleq \sum_{i=1}^n \left(\mathbf{r}'_{r,i} R(\mathbf{r}'_{l,i})\right)$
3. $s_l \triangleq \sum_{i=1}^n ||\mathbf{r}'_{l,i}||_2^2$

Then we can write this objective for the optimal scaling factor s^* as:

$$s^* = \arg\min_s \{J(s) \triangleq s_r - 2sD + s^2 s_l\}$$

Since this is an unconstrained optimization problem, we can solve this by taking the derivative w.r.t. s and setting it equal to 0:

$$\begin{aligned}\frac{dJ(s)}{ds} &= \frac{d}{ds}\left(s_r - 2sD + s^2 s_l\right) = 0\\ &= -2D + s^2 s_l = 0 \implies s = \frac{D}{s_l}\end{aligned}$$

As we also saw with rotation, this does not give us an exact answer without finding the orthonormal matrix R, but now we are able to remove scale factor and back-solve for it later using our optimal rotation.

1.3.2 Issues with Symmetry

Symmetry question: What if instead of going from the left coordinate system to the right one, we decided to go from right to left? In theory, this should be possible: we should be able to do this simply by **negating translation** and **inverting our rotation and scaling terms**. But in general, doing this in practice with our OLS approach above does not lead to $s_{\text{inverse}} = \frac{1}{s}$ - i.e. inverting the optimal scale factor does not give us the scale factor for the reverse problem.

Intuitively, this is the case because the version of OLS we used above "cheats" and tries to minimize error by shriking the scale by more than it should be shrunk. This occurs because it brings the points closer together, thereby minimizing, on average, the error term. Let us look at an alternative formulation for our error term that accounts for this optimization phenomenon.

1.3.3 Solving for Scaling Using Least Squares: Symmetric Case

Let us instead write our objective as:

$$\mathbf{e}_i = \frac{1}{\sqrt{s}}\mathbf{r}'_{r,i} = \sqrt{s}R(\mathbf{r}'_{l,i})$$

Then we can write our objective and optimization problem over scale as:

$$\begin{aligned}s^* &= \arg\min_s \sum_{i=1}^n ||\frac{1}{\sqrt{s}}\mathbf{r}'_{r,i} - \sqrt{s}R(\mathbf{r}'_{l,i})||_2^2\\ &= \arg\min_s \sum_{i=1}^n \left(\frac{1}{s}||\mathbf{r}'_{r,i}||_2^2\right) - 2\sum_{i=1}^n \left(\mathbf{r}'_{r,i}R(\mathbf{r}'_{l,i})\right) + s\sum_{i=1}^n ||R(\mathbf{r}'_{l,i})||_2^2\\ &= \arg\min_s \frac{1}{s}\sum_{i=1}^n \left(||\mathbf{r}'_{r,i}||_2^2\right) - 2\sum_{i=1}^n \left(\mathbf{r}'_{r,i}R(\mathbf{r}'_{l,i})\right) + s\sum_{i=1}^n ||\mathbf{r}'_{l,i}||_2^2 \quad \textbf{(Rotation preserves vector lengths)}\end{aligned}$$

We then take the same definitions for these terms that we did above:

1. $s_r \triangleq \sum_{i=1}^n \left(||\mathbf{r}'_{r,i}||_2^2\right)$
2. $D \triangleq \sum_{i=1}^n \left(\mathbf{r}'_{r,i} R(\mathbf{r}'_{l,i})\right)$
3. $s_l \triangleq \sum_{i=1}^n ||\mathbf{r}'_{l,i}||_2^2$

Then, as we did for the asymmetric OLS case, we can write this objective for the optimal scaling factor s^* as:

$$s^* = \arg\min_s \{J(s) \triangleq \frac{1}{s}s_r - 2D + s s_l\}$$

Since this is an unconstrained optimization problem, we can solve this by taking the derivative w.r.t. s and setting it equal to 0:

$$\frac{dJ(s)}{ds} = \frac{d}{ds}\left(\frac{1}{s}s_r - 2D + ss_l\right) = 0$$
$$= -\frac{1}{s^2}s_r + s_l = 0 \implies s^2 = \frac{s_l}{s_r}$$

Therefore, we can see that going in the reverse direction preserves this inverse (you can verify this mathematically and intuitively by simply setting $\mathbf{r}'_{r,i} \leftrightarrow \mathbf{r}'_{l,i}\ \forall\, i \in \{1, ..., n\}$ and noting that you will get $s^2_{\text{inverse}} = \frac{s_r}{s_l}$). Since this method better preserves symmetry, it is preferred.

Intuition: Since s no longer depends on correspondences (matches between points in the left and right point clouds), then the scale simply becomes the ratio of the point cloud sizes in both coordinate systems (note that s_l and s_r correspond to the summed vector lengths of the centroid-subtracted point clouds, which means they reflect the variance/spread/size of the point cloud in their respective coordinate systems.

We can deal with translation and rotation in a correspondence-free way, while also allowing for us to decouple rotation. Let us also look at solving rotation, which is covered in the next section.

1.4 Solving for Optimal Rotation in Absolute Orientation

Recall for rotation (see lecture 18 for details) that we switched from optimizing over orthonormal rotation matrices to optimizing over quaternions due to the lessened number of optimization constraints that we must adhere to. With our quaternion optimization formulation, our problem becomes:

$$\max_{\overset{o}{q}} \overset{o}{q}^T N \overset{o}{q}, \text{ subject to } \overset{o}{q}^T \overset{o}{q} = 1$$

If this were an unconstrained optimization problem, we could solve by taking the derivative of this objective w.r.t. our quaternion $\overset{o}{q}$ and setting it equal to zero. Note the following helpful identities with matrix and vector calculus:

1. $\frac{d}{d\mathbf{a}}(\mathbf{a} \cdot \mathbf{b}) = \mathbf{b}$
2. $\frac{d}{d\mathbf{a}}(\mathbf{a}^T M \mathbf{b}) = 2M\mathbf{b}$

However, since we are working with quaternions, we must take this constraint into account. We saw in lecture 18 that we did this with using **Lagrange Multiplier** - in this lecture it is also possible to take this specific kind of vector length constraint into account using **Rayleigh Quotients**.

What are Rayliegh Quotients? The intuitive idea behind them: How do I prevent my parameters from becoming too large) positive or negative) or too small (zero)? We can accomplish this by dividing our objective by our parameters, in this case our constraint. In this case, with the Rayleigh Quotient taken into account, our objective becomes:

$$\frac{\overset{o}{q}^T N \overset{o}{q}}{\overset{o}{q}^T \overset{o}{q}} \quad \left(\textbf{Recall that } N \triangleq \sum_{i=1}^{n} R^T_{l,i} R_{r,i}\right)$$

How do we solve this? Since this is now an unconstrained optimization problem, we can solve this simply using the rules of calculus:

$$J(\overset{o}{q}) \triangleq= \frac{\overset{o}{q}^T N \overset{o}{q}}{\overset{o}{q}^T \overset{o}{q}}$$
$$\frac{dJ(\overset{o}{q})}{d\overset{o}{q}} = \frac{d}{d\overset{o}{q}} \frac{\overset{o}{q}^T N \overset{o}{q}}{\overset{o}{q}^T \overset{o}{q}} = 0$$
$$= \frac{\frac{d}{d\overset{o}{q}}(\overset{o}{q}^T N \overset{o}{q})\overset{o}{q}^T \overset{o}{q} - \overset{o}{q}^T N \overset{o}{q} \frac{d}{d\overset{o}{q}}(\overset{o}{q}^T \overset{o}{q})}{(\overset{o}{q}^T \overset{o}{q})^2} = 0$$
$$= \frac{2N\overset{o}{q}}{\overset{o}{q}^T \overset{o}{q}} - \frac{2\overset{o}{q}}{(\overset{o}{q}^T \overset{o}{q})^2}(\overset{o}{q}^T N \overset{o}{q}) = 0$$

From here, we can write this first order condition result as:

$$N\overset{o}{q} = \frac{\overset{o}{q}^T N \overset{o}{q}}{\overset{o}{q}^T \overset{o}{q}} \overset{o}{q}$$

Note that $\frac{\overset{o}{q}^T N \overset{o}{q}}{\overset{o}{q}^T \overset{o}{q}} \in \mathbb{R}$ (this is our objective). Therefore, we are searching for a vector of quaternion coefficients such applying the rotation matrix to this vector simply produces a scalar multiple of it - i.e. an eigenvector of the matrix N. Letting $\lambda \triangleq \frac{\overset{o}{q}^T N \overset{o}{q}}{\overset{o}{q}^T \overset{o}{q}}$, then this simply becomes $N\overset{o}{q} = \lambda \overset{o}{q}$. Since this optimization problem is a maximization problem, this means that we can pick the **eigenvector** of N that corresponds to the **largest eigenvalue** (which in turn maximizes the objective consisting of the Rayleigh quotient $\frac{\overset{o}{q}^T N \overset{o}{q}}{\overset{o}{q}^T \overset{o}{q}}$, which is the eigenvalue.

Even though this quaternion-based optimization approach requires taking this Rayleigh Quotient into account, it is much easier to do this optimization than to solve for orthonormal matrices, which either require a complex Lagrangian (if we solve with Lagrange multipliers) or an SVD decomposition from Euclidean space to the $\mathbf{SO}(3)$ group (which also happens to be a manifold).

This approach raises a few questions:

- How many correspondences are needed to solve these optimization problems? Recall a correspondence is when we say two 3D points in different coordinate systems belong to the same point in 3D space, i.e. the same point observed in two separate frames of reference.
- When do these approaches fail?

We cover these two questions in more detail below.

1.4.1 How Many Correspondences Do We Need?

Recall that we are looking for 6 parameters (for translation and rotation) or 7 parameters (for translation, rotation, and scaling). Since each correspondence provides three constraints (since we equate the 3-dimensional coordinates of two 3D points in space), assuming non-redundancy, then we can solve this with two correspondences.

Let us start with two correspondences: if we have two objects corresponding to the correspondences of points in the 3D world, then if we rotate one object about axis, we find this does not work, i.e. we have an additional degree of freedom. Note that the distance between correspondences is fixed.

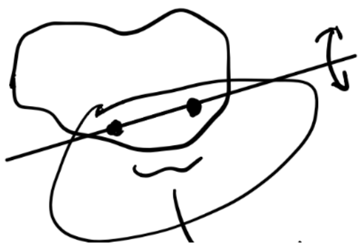


Figure 1: Using two correspondences leads to only satisfying 5 of the 6 needed constraints to solve for translation and rotation between two point clouds.

Because we have one more degree of freedom, this accounts for only 5 of the 6 needed constraints to solve for translation and rotation, so we need to have **at least 3 correspondences**.

With 3 correspondences, we get 9 constraints, which leads to some redundancies. We can add more constraints by incorporating **scaling** and generalizing the allowable transformations between the two coordinate systems to be the **generalized linear transformation** - this corresponds to allowing non-orthonormal rotation transformations. This approach gives us 9 unknowns!

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} a_{14} \\ a_{24} \\ a_{34} \end{bmatrix}$$

But we also have to account for translation, which gives us another 3 unknowns, giving us 12 in total and therefore requiring at least 4 non-redundant correspondences in order to compute the full general linear transformation. Note that this doesn't have any constraints as well!

On a practical note, this is often not needed, especially for finding the absolute orientation between two cameras, because oftentimes the only transformations that need to be considered due to the design constraints of the system (e.g. an autonomous car with two lidar systems, one on each side) are **translation** and **rotation**.

1.4.2 When do These Approaches Fail?

These approaches can fail when we do not have enough correspondences. In this case, the matrix N will become singular, and will produce eigenvalues of zero. A more interesting failure case occurs when the points of one or both of the point clouds are **coplanar**. Recall that we solve for the eigenvalues of a matrix (N in this case) using the characteristic equation given by:

$$\textbf{Characteristic Equation}: \ \det|N - \lambda \mathbf{I}| = 0$$
$$\textbf{Leads to 4th-order polynomial}: \ \lambda^4 + c_3\lambda^3 + c_2\lambda^2 + c_1\lambda + c_0 = 0$$

Recall that our matrix N composed of the data has some special properties:

1. $c_3 = \text{tr}(N) = 0$ (This is actually a great feature, since usually the first step in solving 4th-order polynomial systems is eliminating the third-order term).
2. $c_2 = 2\text{tr}(M^T M)$, where M is defined as the sum of dyadic products between the points in the point clouds:
$$M \triangleq \left(\sum_{i=1}^{n} \mathbf{r}'_{l,i}\mathbf{r}'^{\,T}_{r,i}\right) \in \mathbb{R}^{3\times 3}$$
3. $c_1 = 8\det|M|$
4. $c_0 = \det|N|$

What happens if $\det|M| = 0$, i.e. the matrix M is singular? Then using the formulas above we must have that the coefficient $c_1 = 0$. Then this problem reduces to:

$$\lambda^4 + c_2\lambda^2 + c_0 = 0$$

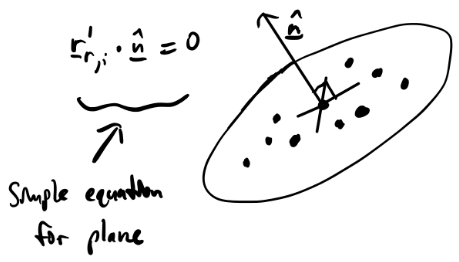
This case corresponds to a special geometric case/configuration of the point clouds - specifically, when points are **coplanar**.

1.4.3 What Happens When Points are Coplanar?

When points are coplanar, we have that the matrix N, composed of the sum of dyadic products between the correspondences in the two point clouds, will be singular.

To describe this plane in space, we need only find a normal vector $\hat{\mathbf{n}}$ that is orthogonal to all points in the point cloud - i.e. the component of each point in the point cloud in the $\hat{\mathbf{n}}$ direction is 0. Therefore, we can describe the plane by the equation:

$$\mathbf{r}'_{r,i} \cdot \hat{\mathbf{n}} = 0 \ \forall \ i \in \{1, ..., n\}$$



$\underline{r}'_{r,i} \cdot \hat{\underline{n}} = 0$
Simple equation for plane
$\hat{\underline{n}}$

Figure 2: A coplanar point cloud can be described entirely by a surface normal of the plane $\hat{\mathbf{n}}$.

Note: In the absence of measurement noise, if one point cloud is coplanar, the the other point cloud must be as well (assuming that the transformation between the point clouds is a linear transformation). This does not necessarily hold when measurement noise is introduced.

Recall that our matrix M, which we used above to compute the coefficients of the characteristic polynomial describing this system, is given by:

$$M \triangleq \sum_{i=1}^{n} \mathbf{r}'_{r,i}{\mathbf{r}'_{l,i}}^T$$

Then, rewriting our equation for the plane above, we have:

$$\begin{aligned}\mathbf{r}'_{r,i} \cdot \hat{\mathbf{n}} = 0 \implies M\hat{\mathbf{n}} &= \left(\sum_{i=1}^{n} \mathbf{r}'_{r,i}{\mathbf{r}'_{l,i}}^T\right) \hat{\mathbf{n}} \\ &= \sum_{i=1}^{n} \mathbf{r}'_{r,i}{\mathbf{r}'_{l,i}}^T \hat{\mathbf{n}} \\ &= \sum_{i=1}^{n} \mathbf{r}'_{r,i} 0 \\ &= 0\end{aligned}$$

Therefore, when a point cloud is coplanar, the **null space** of M is non-trivial (it is given by at least Span($\{\hat{\mathbf{n}}\}$), and therefore M is singular. Recall that a matrix $M \in \mathbb{R}^{n \times d}$ is singular if $\exists\, \mathbf{x} \in \mathbb{R}^d, \mathbf{x} \neq \mathbf{0}$ such that $M\mathbf{x} = \mathbf{0}$, i.e. the matrix has a non-trivial null space.

1.4.4 What Happens When Both Coordinate Systems Are Coplanar

Visually, when two point clouds are coplanar, we have:

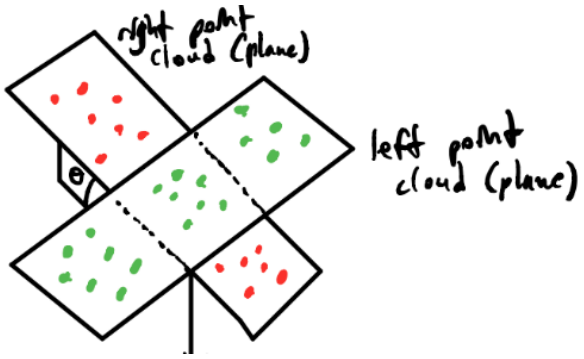


Figure 3: Two coplanar point clouds. This particular configuration allows us to estimate rotation in two simpler steps.

In this case, we can actually decompose finding the right rotation into two simpler steps!

1. Rotate one plane so it lies on top of the other plane. We can read off the **axis** and **angle** from the unit normal vectors of these two planes describing the coplanarity of these point clouds, given respectively by $\hat{\mathbf{n}}_1$ and $\hat{\mathbf{n}}_2$:

   - **Axis**: We can find the axis by noting that the axis vector will be parallel to the cross product of $\hat{\mathbf{n}}_1$ and $\hat{\mathbf{n}}_2$, simply scaled to a unit vector:

   $$\hat{\boldsymbol{\omega}} = \frac{\hat{\mathbf{n}}_1 \times \hat{\mathbf{n}}_2}{||\hat{\mathbf{n}}_1 \times \hat{\mathbf{n}}||_2}$$

   - **Angle**: We can also solve for the angle using the two unit vectors $\hat{\mathbf{n}}_1$ and $\hat{\mathbf{n}}_2$:

   $$\begin{aligned}\cos\theta &= \hat{\mathbf{n}}_1 \cdot \hat{\mathbf{n}}_2 \\ \sin\theta &= \hat{\mathbf{n}}_1 \times \hat{\mathbf{n}}_2 \\ \theta &= \arctan 2\left(\frac{\sin\theta}{\cos\theta}\right)\end{aligned}$$

We now have an axis angle representation for rotation between these two planes, and since the points describe each of the respective point clouds, therefore, a rotation between the two point clouds! We can convert this axis-angle representation into a quaternion with the formula we have seen before:

$$\overset{o}{q} = \left(\cos\frac{\theta}{2}, \sin\frac{\theta}{2}\hat{\boldsymbol{\omega}}\right)$$

2. Perform an in-plane rotation. Now that we have the quaternion representing the rotation between these two planes, we can orient two planes on top of each other, and then just solve a 2D least-squares problem to solve for our in-place rotation.

With these steps, we have a rotation between the two point clouds!

1.5 Robustness

In many methods in this course, we have looked at the use of **Least Squares** methods to solve for estimates in the presence of noise and many data points. Least squares produces an unbiased, minimum-variance estimate if (along with a few other assumptions) the dataset/measurement noise is Gaussian (Gauss-Markov Theorem) [1]. But what if the measurement noise is non-Gaussian? How do we deal with outliers in this case?

It turns out that **Least Squares** methods are not robust to outliers. One alternative approach is to use absolute error instead. Unfortunately, however, using absolute error does not have a closed-form solution. What are our other options for dealing with outliers? One particularly useful alternative is **RANSAC**.

RANSAC, or **Random Sample Consensus**, is an algorithm for robust estimation with **least squares** in the presence of outliers in the measurements. The goal is to find a least squares estimate that includes, within a certain threshold band, a set of inliers corresponding to the inliers of the dataset, and all other points outside of this threshold bands as outliers. The high-level steps of RANSAC are as follows:

1. **Random Sample**: Sample the minimum number of points needed to fix the transformation (e.g. 3 for absolute orientation; some recommend taking more).
2. **Fit random sample of points**: Usually this involves running least squares on the sample selected. This fits a line (or hyperplane, in higher dimensions), to the randomly-sampled points.
3. **Check Fit**: Evaluate the line fitted on the randomly-selected subsample on the rest of the data, and determine if the fit produces an estimate that is consistent with the "inliers" of your dataset. If the fit is good enough accept it, and if it is not, run another sample. Note that this step has different variations - rather than just immediately terminating once you have a good fit, you can run this many times, and then take the best fit from that.

   Furthermore, for step 3, we threshold the band from the fitted line/hyperplane to determine which points of the dataset are inliers, and which are outliers (see figure below). This band is usually given by a 2ϵ band around the fitted line/hyperplane. Typically, this parameter is determined by knowing some intrinsic structure about the dataset.

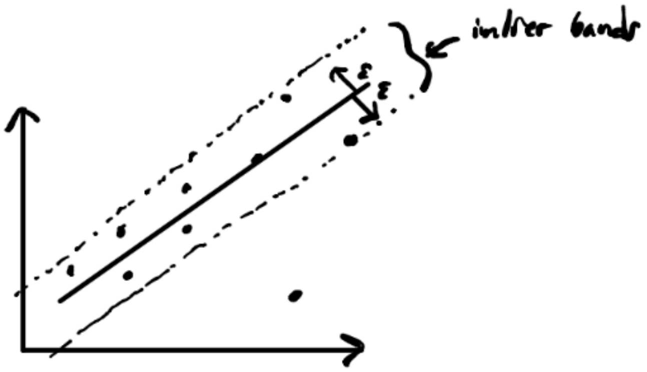


Figure 4: To evaluate the goodness of fit of our sampled points, as well as to determine inliers and outliers from our dataset, we have a 2ϵ thick band centered around the fitted line.

Another interpretation of RANSAC: counting the "maximimally-occupied" cell in Hough transform parameter space! Another way to find the best fitting line that is robust to outliers:

1. Repeatedly sample subsets from the dataset/set of measurements, and fit these subsets of points using least squares estimates.

2. For each fit, map the points to a discretized Hough transform parameter space, and have an accumulator array that keeps track of how often a set of parameters falls into a discretized cell. Each time a set of parameters falls into a discretized cell, increment it by one.

3. After N sets of random samples/least squares fits, pick the parameters corresponding to the cell that is "maximally-occupied", aka has been incremented the most number of times! Take this as your outlier-robust estimate.

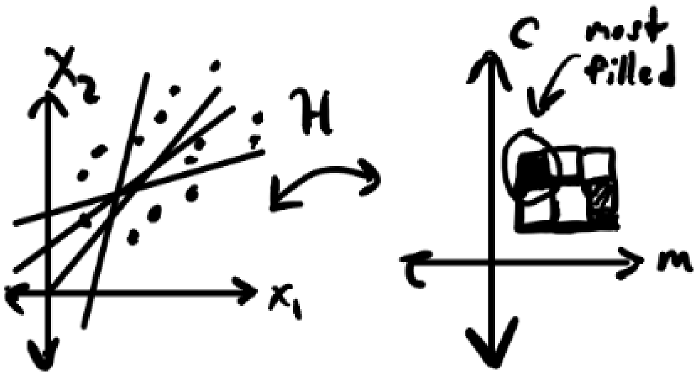


Figure 5: Another way to perform RANSAC using Hough Transforms: map each fit from the subsamples of measurements to a discretized Hough Transform (parameter) space, and look for the most common discretized cell in parameter space to use for an outlier-robust least-squares estimate.

1.6 Sampling Space of Rotations

Next, we will shift gears to discuss the **sampling space of rotations**.

Why are we interested in this space? Many orientation problems we have studied so far do not have a closed-form solution and may require sampling. How do we sample from the space of rotations?

1.6.1 Initial Procedure: Sampling from a Sphere

Let us start by **sampling from a unit sphere** (we will start in 3D, aiming eventually for 4D, but our framework will generalize easily from 3D to 4D). Why a sphere? Recall that we are interested in sampling for the coefficients of a unit quaternion $\overset{o}{q} = (q_0, q_x, q_y, q_z), ||\overset{o}{q}||_2^2 = 1$.

One way to sample from a sphere is with latitude and longitude, given by (θ_i, ϕ_i), respectively. The problem with this approach, however, is that we sample points that are close together at the poles. Alternatively, we can generate random longitude θ_i and ϕ_i, where:

- $-\frac{\pi}{2} \leq \theta_i \leq \frac{\pi}{2} \; \forall \; i$
- $-\pi \leq \phi_i \leq \pi \; \forall \; i$

But this approach suffers from the same problem - it samples too strongly from the poles. Can we do better?

1.6.2 Improved Approach: Sampling from a Cube

To achieve more uniform sampling from a sphere, what if we sampled from a unit cube (where the origin is given the center of the cube), and map the sampled points to an enscribed unit sphere within the cube?

Idea: Map all points (both inside the sphere and outside the sphere/inside the cube) onto the sphere by connecting a line from the origin to the sampled point, and finding the point where this line intersects the sphere.

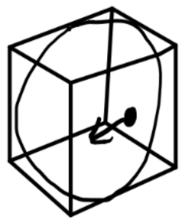


Figure 6: Sampling from a sphere by sampling from a cube and projecting it back to the sphere.

Problem with this approach: This approach disproportionately samples more highly on/in the direction of the cube's edges. We could use sampling weights to mitigate this effect, but better yet, we can simply discard any samples that fall outside the sphere. To avoid numerical issues, it is also best to discard points very close to the sphere.

Generalization to 4D: As we mentioned above, our goal is to generalize this from 3D to 4D. Cubes and spheres simply become 4-dimensional - enabling us to sample quaternions.

1.6.3 Sampling From Spheres Using Regular and Semi-Regular Polyhedra

We saw the approach above requires discarding samples, which is computationally-undesirable because it means we will probabilistically have to generate more samples than if we were able to sample from the sphere alone. To make this more efficient, let us consider shapes that form a "tighter fit" around the sphere - for instance: **polyhedra**! Some polyhedra we can use:

- **Tetrahedra** (4 faces)
- **Hexahedra** (6 faces)
- **Octahedra** (8 faces)
- **Dodecahedra** (12 faces)
- **Icosahedra** (20 faces)

These polyhedra are also known as the **regular solids**.

As we did for the cube, we can do the same for polyhedra: to sample from the sphere, we can sample from the polyhedra, and then **project** onto the point on the sphere that intersects the line from the origin to the sampled point on the polyhedra. From this, we get **great circles** from the edges of these polyhedra on the sphere when we project.

Fun fact: Soccer balls have 32 faces! More related to geometry: soccer balls are part of a group of **semi-regular** solids, specifically an **icosadodecahedron**.

1.6.4 Sampling in 4D: Rotation Quaternions and Products of Quaternions

Now we are ready to apply these shapes for sampling quaternions in 4D. Recall that our goal with this sampling task is to find the rotation between two point clouds, e.g. two objects. We need a uniform way of sampling this spae. We can start with the hexahedron. Below are 10 elementary rotations we use (recall that a quaternion is given in axis-angle notation by $\overset{o}{q} = (\cos\left(\frac{\theta}{2}\right), \sin\left(\frac{\pi}{2}\right)\hat{\boldsymbol{\omega}})$):

1. **Identity rotation**: $\overset{o}{q} = (1, 0)$
2. π **about** $\hat{\mathbf{x}}$: $\overset{o}{q} = (\cos\left(\frac{\pi}{2}\right), \sin\left(\frac{\pi}{2}\right)\hat{\mathbf{x}}) = (0, \hat{\mathbf{x}})$
3. π **about** $\hat{\mathbf{y}}$: $\overset{o}{q} = (\cos\left(\frac{\pi}{2}\right), \sin\left(\frac{\pi}{2}\right)\hat{\mathbf{y}}) = (0, \hat{\mathbf{y}})$
4. π **about** $\hat{\mathbf{z}}$: $\overset{o}{q} = (\cos\left(\frac{\pi}{2}\right), \sin\left(\frac{\pi}{2}\right)\hat{\mathbf{z}}) = (0, \hat{\mathbf{z}})$
5. $\frac{\pi}{2}$ **about** $\hat{\mathbf{x}}$: $\overset{o}{q} = (\cos\left(\frac{\pi}{4}\right), \sin\left(\frac{\pi}{4}\right)\hat{\mathbf{x}}) = \frac{1}{\sqrt{2}}(1, \hat{\mathbf{x}})$
6. $\frac{\pi}{2}$ **about** $\hat{\mathbf{y}}$: $\overset{o}{q} = (\cos\left(\frac{\pi}{4}\right), \sin\left(\frac{\pi}{4}\right)\hat{\mathbf{y}}) = \frac{1}{\sqrt{2}}(1, \hat{\mathbf{y}})$

7. $\frac{\pi}{2}$ **about $\hat{\mathbf{z}}$:** $\overset{o}{q} = (\cos\left(\frac{\pi}{4}\right), \sin\left(\frac{\pi}{4}\right)\hat{\mathbf{z}}) = \frac{1}{\sqrt{2}}(1, \hat{\mathbf{z}})$

8. $-\frac{\pi}{2}$ **about $\hat{\mathbf{x}}$:** $\overset{o}{q} = (\cos\left(-\frac{\pi}{4}\right), \sin\left(-\frac{\pi}{4}\right)\hat{\mathbf{x}}) = \frac{1}{\sqrt{2}}(1, -\hat{\mathbf{x}})$

9. $-\frac{\pi}{2}$ **about $\hat{\mathbf{y}}$:** $\overset{o}{q} = (\cos\left(-\frac{\pi}{4}\right), \sin\left(-\frac{\pi}{4}\right)\hat{\mathbf{y}}) = \frac{1}{\sqrt{2}}(1, -\hat{\mathbf{y}})$

10. $-\frac{\pi}{2}$ **about $\hat{\mathbf{z}}$:** $\overset{o}{q} = (\cos\left(-\frac{\pi}{4}\right), \sin\left(-\frac{\pi}{4}\right)\hat{\mathbf{z}}) = \frac{1}{\sqrt{2}}(1, -\hat{\mathbf{z}})$

These 10 rotations by themselves give us 10 ways to sample the rotation space. How can we construct more samples? We can do so by **taking quaternion products**, specifically, products of these 10 quaternions above. Let us look at just a couple of these products:

1. $(0, \hat{\mathbf{x}})(0, \hat{\mathbf{y}})$:

$$\begin{aligned}(0, \hat{\mathbf{x}})(0, \hat{\mathbf{y}}) &= (0 - \hat{\mathbf{x}} \cdot \hat{\mathbf{y}}, 0\hat{\mathbf{x}} + 0\hat{\mathbf{y}} + \hat{\mathbf{x}} \times \hat{\mathbf{y}}) \\ &= (-\hat{\mathbf{x}} \cdot \hat{\mathbf{y}}, \hat{\mathbf{x}} \times \hat{\mathbf{y}}) \\ &= (0, \hat{\mathbf{z}})\end{aligned}$$

   We see that this simply produces the third axis, as we would expect. This does not give us a new rotation to sample from. Next, let us look at one that does.

2. $\frac{1}{\sqrt{2}}(1, \hat{\mathbf{x}})\frac{1}{\sqrt{2}}(1, \hat{\mathbf{y}})$:

$$\begin{aligned}\frac{1}{\sqrt{2}}(1, \hat{\mathbf{x}})\frac{1}{\sqrt{2}}(1, \hat{\mathbf{y}}) &= \frac{1}{2}(1 - \hat{\mathbf{x}} \cdot \hat{\mathbf{y}}, \hat{\mathbf{y}} + \hat{\mathbf{x}} + \hat{\mathbf{x}} \times \hat{\mathbf{y}}) \\ &= \frac{1}{2}(1, \hat{\mathbf{x}} + \hat{\mathbf{y}} + \hat{\mathbf{x}} \times \hat{\mathbf{y}})\end{aligned}$$

   This yields the following axis-angle representation:

   - Axis: $\frac{1}{\sqrt{3}}(1\ 1\ 1)$
   - Angle: $\cos\left(\frac{\theta}{2}\right) = \frac{1}{2} \implies \frac{\theta}{2} = \frac{\pi}{3} \implies \theta = \frac{2\pi}{3}$

   Therefore, we have produced a new rotation that we can sample from!

These are just a few of the pairwise quaternion products we can compute. It turns out that these pairwise quaternion products produce a total of **24 new rotations** from the original 10 rotations. These are helpful for achieving greater sampling granularity when sampling the rotation space.

1.7 References

1. Gauss-Markov Theorem, https://en.wikipedia.org/wiki/Gauss%E2%80%93Markov_theorem