6.801/6.866: Machine Vision, Lecture 17

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.

1 Lecture 17: Photogrammetry, Orientation, Axes of Inertia, Symmetry, Absolute, Relative, Interior, and Exterior Orientation

This lecture marks a transition in what we have covered so far in the course from lower-level machine vision to higher-level problems, beginning with photogrammetry.

Photogrammetry: "Measurements from Imagery", for example, map-making.

1.1 Photogrammetry Problems: An Overview

Four important problems in photogrammetry that we will cover are:

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

Below we discuss each of these problems at a high level. We will be discussing these problems in greater depth later in this and following lectures.

1.1.1 Absolute Orientation

We will start with covering **absolute orientation**. This problem asks about the relationship between two or more objects (cameras, points, other sensors) in 3D. Some examples of this problem include:

1. Given two 3D sensors, such as lidar (light detection and ranging) sensors, our goal is to find the **transformation**, or **pose**, between these two sensors.
2. Given one 3D sensor, such as a lidar sensor, and two objects (note that this could be two distinct objects at a single point in time, or a single object at two distinct points in time), our goal is to find the **transformation**, or **pose**, between the two objects.

1.1.2 Relative Orientation

This problem asks how we can find the 2D $\longleftrightarrow$ 2D relationship between two objects, such as cameras, points, or other sensors. This type of problem comes up frequently in machine vision, for instance, **binocular stereo**. Two high-level applications include:

1. Given two cameras/images that these cameras take, our goal is to extract 3D information by finding the relationship between two 2D images.
2. Given two cameras, our goal is to find the (relative) **transformation**, or **pose**, between the two cameras.

1.1.3 Exterior Orientation

This photogrammetry problem aims from going 2D $\longrightarrow$ 3D. One common example in robotics (and other field related to machine vision) is **localization**, in which a robotic agent must find their location/orientation on a map given 2D information from a camera (as well as, possibly, 3D laser scan measurements).

More generally, with **localization**, our goal is to find where we are and how we are oriented in space given a 2D image and a 3D model of the world.

1.1.4 Interior Orientation

This photogrammetry problem aims from going 3D $\longrightarrow$ 2D. The most common application of this problem is **camera calibration**. Camera calibration is crucial for high-precision imaging, as well as solving machine and computer vision problems such as Bundle Adjustment [1]. Finding a principal point is another example of the interior orientation problem.

1.2 Absolute Orientation

We will begin our in-depth discussion and analysis of these four problems with **absolute orientation**. Let us start by introducing binocular stereo.

1.2.1 Binocular Stereopsis

To motivate binocular stereo, we will start with a fun fact: Humans have ≈ 12 depth cues. One of these is **binocular stereopsis**, or **binocular stereo** (binocular $\approx$ two sensors).

Binocular stereo is given by the figure below:

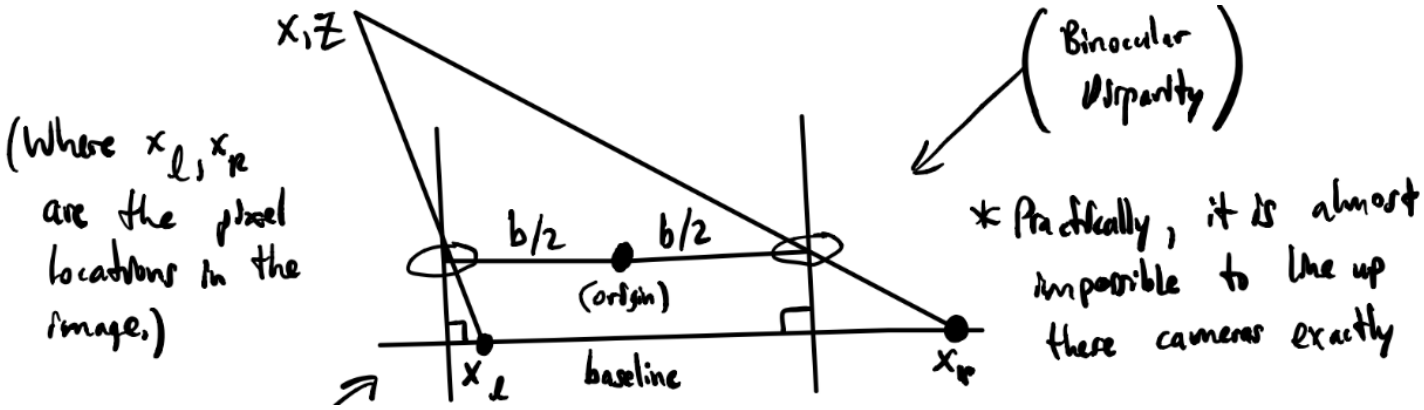

Figure 1: The problem of binocular stereo. Having two 2D sensors enables us to recover 3D structure (specifically, depth) from the scene we image. A few key terms/technicalities to note here: (i) The origin is set to be halfway between the two cameras, (ii) The distance between the cameras is called the **baseline**, and (iii) **binocular disparity** refers to the phenomenon of having each camera generate a different image. Finally, note that in practice, it is almost impossible to line up these cameras exactly.

Goal: Calculate X and Z. This would not be possible with only monocular stereo (monocular $\approx$ one sensor). Using similar triangles, we have:

$$\frac{X - \frac{b}{2}}{Z} = \frac{x_l}{f}$$
$$\frac{X + \frac{b}{2}}{Z} = \frac{x_r}{f}$$
$$\longrightarrow \frac{x_r - x_l}{f} = \frac{b}{z} \implies z = \frac{bf}{x_r - x_l}$$

Where the **binocular disparity** is given by $(x_r - x_l)$. Solving this system of equations becomes:

$$X = b\frac{x_r + x_l}{x_r - x_l},\ Z = bf\frac{1}{x_r - x_l}$$

Note that, more generally, we can calculate in the same way as well! From these equations, we can see that:

- Increasing the baseline increases X and Z (all things equal)
- Increasing the focal length increases Z.

But it is important to also be mindful of system constraints that determine what range, or the exact value, of what these variables can be. For instance, if we have a self-driving car, we cannot simply have the baseline distance between our two cameras be 100 meters, because this would require mounting the cameras 100 meters apart.

1.2.2 General Case

All of the methods we have looked at so far assume we have already found "interesting points", for instance, through feature detection algorithms (also known as "descriptors") such as SIFT, SURF, ORB, or FAST+BRIEF. In the general case, absolute orientation involves having a point in space that is measured by two separate frames of reference (could be a camera, or another sensor), and the goal is to find the **closest approach** between coordinate systems, a.k.a. the shortest line that connects them. We will arbitrarily assign the 3 x 3 matrix $(\mathbf{x}_l, \mathbf{y}_l, \mathbf{z}_l) \in \mathbb{R}^{3\times3}$ to refer to a "lefthanded" coordinate system, and 3 x 3 matrix $(\mathbf{x}_r, \mathbf{y}_r, \mathbf{z}_r) \in \mathbb{R}^{3\times3}$ to refer to a "righthanded" coordinate system, where our goal is to find the transformation between these two coordinate systems. The diagram below illustrates this:

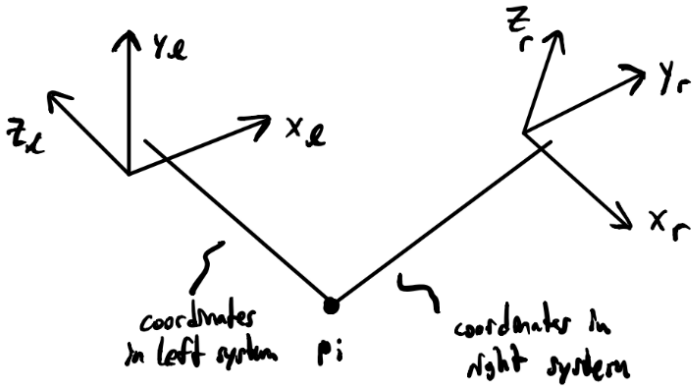

Figure 2: General case of absolute orientation: Given the coordinate systems $(\mathbf{x}_l, \mathbf{y}_l, \mathbf{z}_l) \in \mathbb{R}^{3\times3}$ and $(\mathbf{x}_r, \mathbf{y}_r, \mathbf{z}_r) \in \mathbb{R}^{3\times3}$, our goal is to find the transformation, or pose, between them using points measured in each frame of reference $\mathbf{p}_i$.

Here, we also note the following, as they are important for establishing that we are not limited just to finding the transformation between two camera sensors:

- "Two cameras" does not just mean having two distinct cameras (known as "two-view geometry"); this could also refer to having a single camera with images taken at two distinct points in time (known as "Visual Odometry" or "VO").
- Note also that we could have the same scenario described above when introducing this problem, which is that we have either one camera and multiple objects, or multiple objects and one cameras. Hence, there is a sense of duality here with solving these problems:
  1. One camera, two objects?
  2. Two cameras, one object (Two-View Geometry)?
  3. Camera moving (VO)?
  4. Object moving?

To better understand this problem, it is first important to precisely define the **transformation** or **pose** (translation + rotation) between the two cameras.

1.2.3 Transformations and Poses

Recall that in 3D, we have 6 degrees of freedom (DOF), of which **3** are for **translation**, and **3** are for **rotation**. In the general case of a d-dimensional system, we will have d **translational** degrees of freedom, and $\frac{d(d-1)}{2}$ **rotational** degrees of freedom (which, interestingly, is also equal to $\sum_{i=0}^{d-1} i$. Therefore, for a d-dimensional system, the total degrees of freedom from translation and rotation:

$$\mathbf{DOF}_{\mathrm{T,\ R}}(d) = d + \frac{d(d-1)}{2}$$

What form does our transformation take? Since we are considering these transformations as a transformation determined by translation and rotation, then we can write our transformation as such:

$$\mathbf{r}_r = R(\mathbf{r}_l) + \mathbf{r}_0$$

Where:

- R describes the rotation. Note that this is not necessarily always an orthonormal rotation matrix, and we have more generally parameterized it as a function.
- $\mathbf{r}_0 \in \mathbb{R}^3$ describes the translation.
- The translation vector and the rotation function comprise our unknowns.

When R is described by an orthonormal rotation matrix $\mathbf{R}$, then we require this matrix to have the following properties:

1. $\mathbf{R}^T\mathbf{R} = \mathbf{R}\mathbf{R}^T = I$, i.e. $\mathbf{R}^T = \mathbf{R}^{-1}$
2. $\mathbf{R}$ is skew-symmetric, so we end up having 3 unknowns instead of 9. Skew-symmetric matrices take the form:

$$\mathbf{R} = \begin{bmatrix} a & b & c \\ -b & d & e \\ -c & -e & f \end{bmatrix} \in \mathbb{R}^{3\times 3}$$

3. $\det |R| = +1$ (This constraint is needed to eliminate reflections.)
4. $\mathbf{R} \in \mathbf{SO}(3)$ (Known as the Special Orthogonal group). One note about this: optimization over the Special Orthogonal group (e.g. for estimating optimal rotation matrices) can be difficult, and one way that this can be done is via Singular Value Decomposition (SVD) from this group/manifold onto some Euclidean space $\mathbb{R}^d$ where $d < 3$.

Supposing we have clusters of points from the left and righthand side, and we want to align clusters of points as closely as possible. With a mechanical spring analog, we have the force (given by Hooke's law), and consequently the energy:

$$F = Ke \tag{1}$$

$$E = \int F = \frac{1}{2}ke^2 \tag{2}$$

Where e (the "error") is the distance between the measured point in the two frames of reference. Therefore, the solution to this problem involves "minimizing the energy" of the system. Interestingly, energy minimization is analogous to least squares regression.

Using the definition of our transformation, our error for the i$^{\text{th}}$ point is given by:

$$\mathbf{e}_i = (R(\mathbf{r}_{l,i}) + \mathbf{r}_0) - \mathbf{r}_{r,i}$$

Then our objective for energy minimization and least squares regression is given by:

$$\min_{R,\mathbf{r}_0} \sum_{i=1}^{N} ||r_i||_2^2$$

This is another instance of solving what is known as the "inverse" problem. The "forward" and "inverse" problems are given by:

- **Forward Problem**: $R, \mathbf{r}_0 \longrightarrow \{\mathbf{r}_{r,i}, \mathbf{r}_{l,i}\}_{i=1}^N$ **(Find correspondences)**
- **Inverse Problem**: $\mathbf{r}_{l,i}\}_{i=1}^N \longrightarrow R, \mathbf{r}_0$ **(Find transformation)**

Now that we have framed our optimization problem, can we decouple the optimization over translation and rotation? It turns out we can by setting an initial reference point. For this, we consider two methods.

1.2.4 Procedure - "Method 1"

:

1. Pick a measured point from one set of measured points as the origin.
2. Take a second point, look at the distance between them, and compute the unit vector. Take unit vector as one axis of the system:

$$\mathbf{x}_l = \mathbf{r}_{l,2} - \mathbf{r}_{l,1} \longrightarrow \hat{\mathbf{x}}_l = \frac{\mathbf{x}_l}{||\mathbf{x}_l||_2}$$

3. Take a third vector, and compute the component of the vector that is equal to the vector from point 1 to point 2. Now, points 1, 2, and 3 from (x, y) plane, and the removed component from point 3 forms the y-component of the coordinate system.
4. We can compute the $\mathbf{y}$ vector:

$$\mathbf{y} = (\mathbf{r}_{l,3} - \mathbf{r}_{l,1}) - (\mathbf{r}_{l,3} - \mathbf{r}_{l,1}) \cdot \hat{\mathbf{x}}_l$$

   From here, we then take the unit vector $\mathbf{y}_l = \frac{\mathbf{y}_l}{||\mathbf{y}_l||_2}$. From this, we know $\hat{\mathbf{x}}_l \cdot \hat{\mathbf{y}}_l = 0$.

5. Obtain the z-axis by the cross product:

$$\hat{\mathbf{z}}_l = \hat{\mathbf{x}}_l \times \hat{\mathbf{y}}_l$$

   (Then we also have that $\hat{\mathbf{z}}_l \cdot \hat{\mathbf{y}}_l = 0$ and $\hat{\mathbf{z}}_l \cdot \hat{\mathbf{x}}_l = 0$. This then defines a coordinate system $(\hat{\mathbf{x}}_l, \hat{\mathbf{y}}_l, \hat{\mathbf{z}}_l)$ for the left camera/point of reference. Note that this only requires 3 points!

6. To calculate this for the righthand frame of reference, we can repeat steps 1-5 for the righthand side to obtain the coordinate system $(\hat{\mathbf{x}}_r, \hat{\mathbf{y}}_r, \hat{\mathbf{z}}_r)$.

From here, all we need to do is find the transformation (rotation, since we have artificially set the origin) between the coordinate system $(\hat{\mathbf{x}}_r, \hat{\mathbf{y}}_r, \hat{\mathbf{z}}_r)$ and $(\hat{\mathbf{x}}_l, \hat{\mathbf{y}}_l, \hat{\mathbf{z}}_l)$. Mathematically, we have the following equations:

$$\begin{aligned}
\hat{\mathbf{x}}_r &= R(\hat{\mathbf{x}}_l) \\
\hat{\mathbf{y}}_r &= R(\hat{\mathbf{y}}_l) \\
\hat{\mathbf{z}}_r &= R(\hat{\mathbf{z}}_l)
\end{aligned}$$

We can condense these equations into a matrix equation, and subsequently a matrix inversion problem:

$$\begin{bmatrix} \hat{\mathbf{x}}_r & \hat{\mathbf{y}}_r & \hat{\mathbf{z}}_r \end{bmatrix} = R \begin{bmatrix} \hat{\mathbf{x}}_l & \hat{\mathbf{y}}_l & \hat{\mathbf{z}}_l \end{bmatrix}$$

Then we can solve this as a matrix inversion problem:

$$R = \begin{bmatrix} \hat{\mathbf{x}}_r & \hat{\mathbf{y}}_r & \hat{\mathbf{z}}_r \end{bmatrix} \begin{bmatrix} \hat{\mathbf{x}}_l & \hat{\mathbf{y}}_l & \hat{\mathbf{z}}_l \end{bmatrix}^{-1}$$

A few notes on this system:

- Is this system invertible? Yes, because by construction these vectors are orthogonal to one another.
- 3 vector equalities $\longrightarrow$ 9 scalar equalities
- Are we using all of these constraints?
  - For point 1, we use all 3 constraints.
  - For point 2, we use 2 constraints.
  - For point 3, we use 1 constraint.

  It turns out this system "favors" points over one another (usually sub-optimal).
- This is a somewhat ad hoc method - this could be useful for deriving a fast approximation or initial estimate, but this method is definitely suboptimal.

1.2.5 Procedure - "Method 2"

For this algorithm, recall our prior work with **blob analysis**. We can look at axes of inertia (see the figure below), where the inertia of the "blob" is given by:

$$I = \iint_O r^2 dm$$

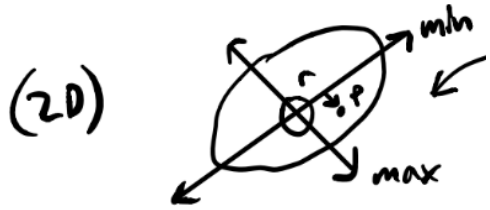

Figure 3: Computing the axes of inertia in a 2D blob.

How do we generalize this from 2D to 3D? How do we figure out these axes of inertia (see the example below)?

$$I = \iiint_O r^2 dm$$

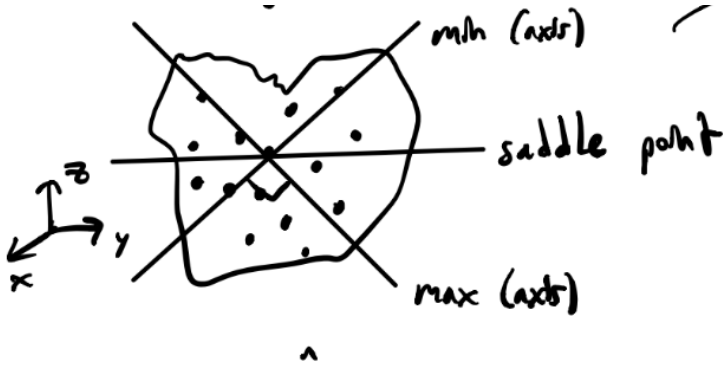

Figure 4: Computing the axes of inertia for a 3D blob - we can generalize the notion of inertia from 2D to 3D.

One trick we can use here is using the **centroid** as the origin.

Using the triangle figure below:

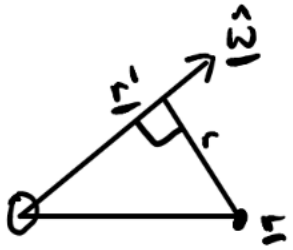

Figure 5: Triangle for computing our $\mathbf{r}'$ vector.

Then we can compute the hypotenuse $\mathbf{r}'$ vector in the $\hat{\boldsymbol{\omega}}$ direction:

$$\mathbf{r}' = (\mathbf{r} \cdot \hat{\boldsymbol{\omega}})\hat{\boldsymbol{\omega}}$$

Then:

$$\mathbf{r} - \mathbf{r}' = \mathbf{r} - (\mathbf{r} \cdot \hat{\boldsymbol{\omega}})\hat{\boldsymbol{\omega}}$$

And we can compute $\mathbf{r}^2$:

$$\begin{aligned}
\mathbf{r}^2 &= (\mathbf{r} - \mathbf{r}') \cdot (\mathbf{r} - \mathbf{r}') \\
&= \mathbf{r} \cdot \mathbf{r} - 2(\mathbf{r} \cdot \hat{\boldsymbol{\omega}})^2 + (\mathbf{r} \cdot \hat{\boldsymbol{\omega}})^2(\hat{\boldsymbol{\omega}} \cdot \hat{\boldsymbol{\omega}}) \\
&= \mathbf{r} \cdot \mathbf{r} - 2(\mathbf{r} \cdot \hat{\boldsymbol{\omega}})^2 + (\mathbf{r} \cdot \hat{\boldsymbol{\omega}})^2 \\
&= \mathbf{r} \cdot \mathbf{r} - (\mathbf{r} \cdot \hat{\omega})^2
\end{aligned}$$

Substituting this into our inertia expression:

$$I = \iiint_O (\mathbf{r} \cdot \mathbf{r} - (\mathbf{r} \cdot \hat{\boldsymbol{\omega}})^2) dm$$

Let us simplify some of these formulas:

1. $(\mathbf{r} \cdot \hat{\boldsymbol{\omega}})^2$:

$$\begin{aligned}
(\mathbf{r} \cdot \hat{\boldsymbol{\omega}})^2 &= (\mathbf{r} \cdot \hat{\boldsymbol{\omega}})(\mathbf{r} \cdot \hat{\boldsymbol{\omega}}) \\
&= (\hat{\boldsymbol{\omega}} \cdot \mathbf{r})(\mathbf{r}\hat{\boldsymbol{\omega}}) \\
&= \hat{\boldsymbol{\omega}}^T(\mathbf{r}\mathbf{r}^T)\hat{\boldsymbol{\omega}}
\end{aligned}$$

   Where $\mathbf{r}\mathbf{r}^T \in \mathbb{R}^{3\times 3}$ is the dyadic product.

2. $(\mathbf{r} \cdot \mathbf{r})$:

$$\begin{aligned}
(\mathbf{r} \cdot \mathbf{r}) &= (\mathbf{r} \cdot \mathbf{r})(\hat{\boldsymbol{\omega}} \cdot \hat{\boldsymbol{\omega}}) \\
&= (\mathbf{r} \cdot \mathbf{r})\hat{\boldsymbol{\omega}}^T\mathbf{I}_3\hat{\boldsymbol{\omega}}
\end{aligned}$$

   Where $\mathbf{I}_3$ is the 3×3 identity matrix.

Therefore, the inertia matrix becomes:

$$\begin{aligned}
I &= \iiint_O (\mathbf{r} \cdot \mathbf{r} - (\mathbf{r} \cdot \hat{\boldsymbol{\omega}})^2) dm \\
&= \iiint_O (\mathbf{r} \cdot \mathbf{r} - (\mathbf{r} \cdot \hat{\boldsymbol{\omega}})^2) dm \\
&= \iiint_O (\mathbf{r} \cdot \mathbf{r})\hat{\boldsymbol{\omega}}^T\mathbf{I}_3\hat{\boldsymbol{\omega}} - \hat{\boldsymbol{\omega}}^T(\mathbf{r}\mathbf{r}^T)\hat{\boldsymbol{\omega}}) dm \\
&= \hat{\boldsymbol{\omega}}^T\Big(\iiint_O ((\mathbf{r} \cdot \mathbf{r})\mathbf{I}_3 - \mathbf{r}\mathbf{r}^T) dm\Big)\hat{\boldsymbol{\omega}}
\end{aligned}$$

From this expression, we want to find the extrema. We can solve for the extrema by solving for the minimium and maximums of this objective:

1. **Minimum**: $\min_{\hat{\boldsymbol{\omega}}} \hat{\boldsymbol{\omega}}^T A \hat{\boldsymbol{\omega}}$
2. **Maximum**: $\max_{\hat{\boldsymbol{\omega}}} \hat{\boldsymbol{\omega}}^T A \hat{\boldsymbol{\omega}}$

Where $A \triangleq \iiint_O ((\mathbf{r} \cdot \mathbf{r})\mathbf{I}_3 - \mathbf{r}\mathbf{r}^T) dm$. This matrix is known as the "inertia matrix".

How can we solve this problem? We can do so by looking for the **eigenvectors** of the inertia matrix:

- For minimization, the **eigenvector** corresponding to the **smallest eigenvalue** of the inertia matrix corresponds to our solution.
- For maximization, the **eigenvector** corresponding to the **largest eigenvalue** of the inertia matrix corresponds to our solution.
- For finding the saddle point, our solution will be the **eigenvector** corresponding to the **middle eigenvalue**.

Since this is a polynomial system of degree 3, we have a closed-form solution! These three eigenvectors will form a coordinate system for the lefthand system.

Taking a step back, let us look at what we have done so far. We have taken the cloud of points from the left frame of reference/coordinate system and have estimated a coordinate system for it by finding an **eigenbasis** from solving these optimization problems over the objective $\hat{\boldsymbol{\omega}}^T A \hat{\boldsymbol{\omega}}$. With this, we can then repeat the same process for the righthand system.

Why does this approach work better?

- We use all the data in both point clouds of 3D data.
- We weight all the points equally, unlike in teh previous case.

But why is this approach not used in practice?

- This approach fails under symmetry - i.e. of the inertia is hte same in all directions (for shapes such as spheres, polyhedra, octahedra, cube, etc.)
- "Double-Edged Sword" - using correspondences can provide you with more information, but can run into issues with symmetry.

1.2.6 Computing Rotations

Next, we will dive into how we can compute and find rotations. To do this, let us first go over the following properties of rotations:

1. **Rotations preserve dot products**: $R(\mathbf{a}) \cdot R(\mathbf{b}) = \mathbf{a} \cdot \mathbf{b}$
2. **Rotations preserve length**: $R(\mathbf{a}) \cdot R(\mathbf{a}) = \mathbf{a} \cdot \mathbf{a}$
3. **Rotations preserve angles**: $|R(\mathbf{a}) \times R(\mathbf{b})| = |\mathbf{a} \times \mathbf{b}|$
4. **Rotations preserve triple products**: $[R(\mathbf{a})\ R(\mathbf{b})\ R(\mathbf{c})] = [\mathbf{a}\ \mathbf{b}\ \mathbf{c}]$ (Where the triple product $[\mathbf{a}\ \mathbf{b}\ \mathbf{c}] = \mathbf{a} \cdot (\mathbf{b} \times \mathbf{c})$).

Using these properties, we are now ready to set this up as least squares, using correspondences between points measured between two coordinate systems:

Transform: $\mathbf{r}_r = R(\mathbf{r}_l) + \mathbf{r}_0$

Error: $\mathbf{e}_i = \mathbf{r}_{r,i} - R(\mathbf{r}_{l,i}) - \mathbf{r}_0$

And we can write our **optimization** as one that minimizes this error term:

$$R^*, \mathbf{r}_0^* \sum_{i=1}^{N} ||\mathbf{e}_i||^2$$

Next, we can compute the centroids of the left and right systems:

$$\bar{\mathbf{r}}_l = \frac{1}{N} \sum_{i=1}^{N} \mathbf{r}_{l,i},\ \bar{\mathbf{r}}_r = \frac{1}{N} \sum_{i=1}^{N} \mathbf{r}_{r,i}$$

We can use these computed centroids from points so we do not have to worry about translation. A new feature of this system is that the new centroid is at the origin. To prove this, let us "de-mean" (subtract the mean) of our coordinates in the left and righthand coordinate systems:

$$\mathbf{r}'_{l,i} = \mathbf{r}_{l,i} - \bar{\mathbf{r}}_l,\ \mathbf{r}'_{r,i} = \mathbf{r}_{r,i} - \bar{\mathbf{r}}_r$$

Because we subtract the mean, the mean of these new points now becomes zero:

$$\hat{\mathbf{r}}'_l = \frac{1}{N} \sum_{i=1}^{N} \mathbf{r}'_{l,i} = 0 = \hat{\mathbf{r}}'_r = \frac{1}{N} \sum_{i=1}^{N} \mathbf{r}'_{r,i}$$

Substituting this back into the objective, we can solve for an optimal rotation R:

$$\begin{aligned} R^* &= \min_{R,\mathbf{r}'_0} \sum_{i=1}^{N} ||\mathbf{r}'_i - R(\mathbf{r}'_{l,i} - \mathbf{r}'_0)||_2^2 \\ &= \min_{R,\mathbf{r}'_0} \sum_{i=1}^{N} ||\mathbf{r}'_i - R(\mathbf{r}'_{l,i})||_2^2 - 2\mathbf{r}'_0 \sum_{i=1}^{N} (\mathbf{r}'_{r,i} - R(\mathbf{r}_{l,i})) + N||\mathbf{r}'_0||_2^2 \\ &= \min_{R,\mathbf{r}'_0} \sum_{i=1}^{N} ||\mathbf{r}'_i - R(\mathbf{r}'_{l,i})||_2^2 + N||\mathbf{r}'_0||_2^2 \end{aligned}$$

Since only the last term depends on $\mathbf{r}'_0$, we can set $\mathbf{r}'_0$ to minimize. Moreover, we can solve for the true $\mathbf{r}_0$ by back-solving later:

$$\mathbf{r}_0 = \bar{\mathbf{r}}_r - R(\bar{\mathbf{r}}_l)$$

Intuitively, this makes sense: the translation vector between these two coordinate systems/point clouds is the difference between the centroid of the right point cloud and the centroid of the left point cloud after it has been rotated.

Since we now have that $\mathbf{r}'_0 = \mathbf{0} \in \mathbb{R}^3$, we can write our error term as:

$$\mathbf{e}_i = \mathbf{r}'_{r,i} - R(\mathbf{r}'_{l,i})$$

Which in turn allows us to write the objective as:

$$\begin{aligned} \min \sum_{i=1}^{N} ||\mathbf{e}_i||_2^2 &= \sum_{i=1}^{N} (\mathbf{r}'_{r,i} - R(\mathbf{r}'_{l,i})(\mathbf{r}'_{r,i} - R(\mathbf{r}'_{l,i})) \\ &= \sum_{i=1}^{N} ||\mathbf{r}'_{r,i}||_2^2 - \sum_{i=1}^{N} (\mathbf{r}'_{r,i} - R(\mathbf{r}'_{l,i})) - \sum_{i=1}^{N} ||\mathbf{r}'_{l,i}||_2^2 \end{aligned}$$

Where the first of these terms is fixed, the second of these terms is to be maximized (since there is a negative sign in front of this term, this thereby minimizes the objective), and the third of these terms is fixed. Therefore, our rotation problem can be simplified to:

$$\min \sum_{i=1}^{N} ||\mathbf{e}_i||_2^2 = -2 \sum_{i=1}^{N} (\mathbf{r}'_{r,i} - R(\mathbf{r}'_{l,i})) \tag{3}$$

To solve this objective, we could take the derivative $\frac{d}{dR}(\cdot)$ of the objective, but constraints make this optimization problem difficult to solve (note that we are not optimizing over a Euclidean search space - rather, we are optimizing over a generalized transformation). We will look into a different representation of R for next class to find solutions to this problem!

1.3 References

1. Bundle Adjustment, https://en.wikipedia.org/wiki/Bundle adjustment

6.801 / 6.866 Machine Vision
Fall 2020