Derivation, Least Squares for Noisy Bunny Point Alignment

We will derive a linear system using least squares, that will find the "best" 4x4 matrix

$$\underline{\underline{M}}$$

that will align N points (using translation and a rotation-like linear transform) of a given dataset, an "original" dataset, of 3D points, as 4x1 vectors of (little) vector

$$\vec{b}^k \qquad (k = 1 \ldots N)$$

into the best match for a rotated, noisy version of the same dataset of 3D points, also 4x1 vectors, which we'll call (big) vector

$$\vec{B}^k \qquad (k = 1 \ldots N).$$

In addition, to use cuBLAS more effectively, at times we will be changing notation, where a summation of indexed items can be turned into matrix operations, using matrix notation instead of summation notation and vice versa.

As a "Rosetta stone" like reference for this, we can relate the i-th element of column vector $\vec{y}$, which we're letting to be a nonsquare matrix $\underline{\underline{A}}$ times **column vector** $\vec{x}$, to be governed by the following matrix and summation relations.

$$\vec{y} = \underline{\underline{A}}\ \vec{x}$$

or

$$y_i = \sum_{j=1}^{n} A_{ij} x_j,\ i = 1 \ldots m$$

Note that the *second* subscript of matrix $\underline{\underline{A}}$ goes with the subscript of column vector $\vec{x}$, and the *first* subscript of $\underline{\underline{A}}$ matches the subscript of column vector $\vec{y}$.

Likewise, if in matrix form,

$$\underline{\underline{C}} = \underline{\underline{A}}\ \underline{\underline{B}}$$

then in summation form,

$$C_{ij} = \sum_{p=1}^{n} A_{ip}\ B_{pj}$$

Note that the *second* subscript of matrix $\underline{\underline{A}}$ goes with the *first* subscript of matrix $\underline{\underline{B}}$ in the summation, and that the two subscripts of matrix $\underline{\underline{C}}$ match the first and last subscripts in the summation expression.

This type of Rosetta-stone-like convention will let you bounce back and forth between matrix and summation conventions, without (as many) transpose or other types of indexing problems, which can help you relate mathematical matrix notations and summations, to use in cuBLAS.

Back to the datasets.

The points are assumed to already be in correspondence. Each point in the original dataset $\vec{b}$ and the noisy transformed version $\vec{B}$ is a 4x1 column vector of the form (little) vector

$$\vec{b} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix},\ \text{and}$$

likewise, each point in the noisy, rotated dataset is in the same form, (big) vector

$$\vec{B} = \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

So we're trying to find the "best" 4x4 matrix $\underline{\underline{M}}$, balanced over all of the points, such that

$$\underline{\underline{M}}\ \vec{b}^k \approx \vec{B}^k$$

The 4x4 matrix $\underline{\underline{M}}$ is assumed to have translation and a linear transform like rotation, but no perspective transformation.

As a result the matrix $\underline{\underline{M}}$ is of the form:

$$\underline{\underline{M}} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & t_x \\ m_{21} & m_{22} & m_{23} & t_y \\ m_{31} & m_{32} & m_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where there are 12 independent variables.

Note that if we multiply it out,

$$\underline{\underline{M}} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha_x + t_1 \\ \alpha_y + t_2 \\ \alpha_z + t_3 \\ 1 \end{pmatrix}$$

.

so we are transforming $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$ by the 3x3 part of $\underline{\underline{M}}$ to get $\begin{pmatrix} \alpha_x \\ \alpha_y \\ \alpha_z \end{pmatrix}$, and then adding a translation vector, $\begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix}$ for the total transformation.

Now we are going to rename the variables in matrix $\underline{\underline{M}}$ so that they are more easily indexed. So for us, now,

$$\underline{\underline{M}} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where there are 12 independent variables, i.e., 12 unknown values of m_{ij}, where $i = 1 \ldots 3$, and $j = 1 \ldots 4$.

With least squares, we create a net, total error term E of squared differences, and then will minimize this by taking the partial derivative with respect to the 12 unknowns.

If the alignment were perfect, then we would find that for each k (and the same matrix $\underline{\underline{M}}$), that the transformed original points would exactly match the target points.

$$\underline{\underline{M}}\ \vec{b}^k = \vec{B}^k$$

or would find that

$$\underline{\underline{M}}\ \vec{b}^k - \vec{B}^k = 0$$

But due to the noise terms, we are expecting the alignment NOT to be perfect, so we will define a "k-th" difference vector,

$$\vec{d}^{\,k} = \underline{\underline{M}}\ \vec{b}^k - \vec{B}^k$$

The magnitude squared of this difference vector is the vector dotted with itself, for each k.

$$|\vec{d}^{\,k}|^2 = \vec{d}^{\,k} \cdot \vec{d}^{\,k}$$

When summed this forms an **objective function** "E" that measures the net error for us to minimize. So we add up all of these magnitude squared values over k, to get a net error "E."

$$E = \sum_{k=1}^{N} \vec{d}^{\,k} \cdot \vec{d}^{\,k}$$

The error "E" is a function of the twelve unknowns, the twelve values of m_{ij}, where $i = 1 \ldots 3$, and $j = 1 \ldots 4$.

Before we take derivatives, let's expand out the dot product in terms of summations.

$$E = \sum_{k=1}^{N} \sum_{i=1}^{4} d_i{}^k\ d_i{}^k$$

To find the optimal values, we take partial derivatives of "E" with respect to each of these twelve unknowns $m_{\ell r}$, where $\ell = 1 \ldots 3$, and $r = 1 \ldots 4$. This is also the gradient of "E." Then we set the result to zero and solve. We could also use Deep Learning Frameworks like **Theano** or **Tensorflow** to take the derivatives for us!

There will be many zero derivatives, since the datapoints themselves don't depend on the unknowns $m_{\ell r}$. The $m_{\ell r}$ factors are twelve **independent variables**, so when they are different variables, the partial derivative of one with respect to another is zero, and when they match, then the partial derivative is 1.

The twelve partial derivatives of E become a linear system to find the twelve unknown values of $m_{\ell r}$. We will also convert from summation form, to "matrix" form, to use for cuBLAS.

So we take partial derivatives of E with respect to $m_{\ell r}$ and set to zero, and then solve.

$$\frac{\partial E}{\partial\ m_{\ell r}} = 2 \sum_{k=1}^{N} \sum_{i=1}^{4} (\frac{\partial\ d_i{}^k}{\partial\ m_{\ell r}}) d_i{}^k$$

where $\ell = 1 \ldots 3$, and $r = 1 \ldots 4$.
Now, the partial derivative of the k-th difference vector is given by:

$$\frac{\partial\ \vec{d}^{\,k}}{\partial\ m_{\ell r}} = \frac{\partial}{\partial\ m_{\ell r}} (\underline{\underline{M}}\ \vec{b}^k - \vec{B}^k)$$

Let's consider the i-th element of both sides, using the "Rosetta Stone" approach for the matrix multiplication to turn into a summation.

$$\frac{\partial\ d_i{}^k}{\partial\ m_{\ell r}} = \frac{\partial}{\partial\ m_{\ell r}} (\sum_{p=1}^{4} m_{ip}\ b_p{}^k - B_i{}^k)$$

$$= \sum_{p=1}^{4} (\frac{\partial}{\partial\ m_{\ell r}} m_{ip})\ b_p{}^k$$

$$= \sum_{p=1}^{4} (\delta_{i\ell}\delta_{pr})\ b_p{}^k$$

where $\delta_{ij} = 1$, if $i = j$, and 0, if $i \neq j$.

These elements of matrix m are independent variables, so a partial derivative of one with respect to another is either zero or 1.

Then, as the next step, the p's don't contribute unless $p = r$, so the sum goes away.

$$\frac{\partial\ d_i{}^k}{\partial\ m_{\ell r}} = \delta_{i\ell}\ b_r{}^k$$

So the partial derivatives of E become:

$$\frac{\partial E}{\partial\ m_{\ell r}} = 2\sum_{k=1}^{N}\sum_{i=1}^{4}(\delta_{i\ell}\ {b_r}^k){d_i}^k$$

But the i's don't contribute to the sum unless $\ell = i$, so that sum goes away too!

$$\frac{\partial E}{\partial\ m_{\ell r}} = 2\sum_{k=1}^{N}({b_r}^k){d_\ell}^k$$

Let's set to zero and divide by two.

$$\sum_{k=1}^{N}({b_r}^k){d_\ell}^k = 0$$

Now let's substitute in the definition of the difference vector ${d_\ell}^k$, where

$${d_\ell}^k = (\underline{\underline{M}}\ \vec{b}^k - \vec{B}^k)_\ell$$

$$= (\sum_{q=1}^{4} m_{\ell q}\ {b_q}^k) - {B_\ell}^k$$

So the linear system for finding the unknown m's is

$$\sum_{k=1}^{N}{b_r}^k\ (\sum_{q=1}^{4} m_{\ell q}\ {b_q}^k - {B_\ell}^k) = 0$$

where $\ell = 1\ldots3$, and $r = 1\ldots4$.

Rearranging,

$$\sum_{k=1}^{N}\sum_{q=1}^{4}{b_r}^k(m_{\ell q}\ {b_q}^k) - \sum_{k=1}^{N}{b_r}^k{B_\ell}^k = 0$$

or

$$\boxed{\sum_{k=1}^{N}\sum_{q=1}^{4}({b_r}^k\ {b_q}^k)m_{\ell q} = \sum_{k=1}^{N}{b_r}^k{B_\ell}^k}$$

where $\ell = 1\ldots3$, and $r = 1\ldots4$, and the m's are the unknowns.

This boxed equation is the main linear system equation that can be massaged into matrix form, to send to cuBLAS, to find the twelve unknown values of $\underline{\underline{M}}$, $m_{\ell q}$.

To see more clearly what is going on, let's expand out the three ℓ cases, $\ell = 1\ldots3$.

For $\ell = 1$:

$$\sum_{k=1}^{N}\sum_{q=1}^{4}({b_r}^k\ {b_q}^k)m_{1q} = \sum_{k=1}^{N}{b_r}^k{B_1}^k$$

This is a linear system for the first four values of matrix $\underline{\underline{M}}$, m_{1q}. Note the ${B_1}^k$ term on the right hand side.

For $\ell = 2$:

$$\sum_{k=1}^{N}\sum_{q=1}^{4}({b_r}^k\ {b_q}^k)m_{2q} = \sum_{k=1}^{N}{b_r}^k{B_2}^k$$

This is a linear system for the next four values of matrix $\underline{\underline{M}}$, m_{2q}.

Note that we have the same linear system matrix on the left hand side made from the $\vec{b}^k$ vectors, but different values on the right hand side, for this linear system. It means that with cuBLAS we can solve all of the systems at once, like the homework suggests.

For $\ell = 3$:

$$\sum_{k=1}^{N}\sum_{q=1}^{4}({b_r}^k\ {b_q}^k)m_{3q} = \sum_{k=1}^{N}{b_r}^k{B_3}^k$$

This is a linear system to find the final four values of matrix $\underline{\underline{M}}$, m_{3q}.

Again, we have the same linear system matrix made from known values of the $\vec{b}^k$ vectors on the left hand side, but different known values on the right hand side, for this linear system, and different unknowns to solve for.

The cuBLAS libray will be able to solve all of these linear systems simultaneously, since we're using the same matrix for the linear system, but different unknowns and different right hand sides.

As the boxed equation suggests, these three equations can be converted into one giant matrix system, for use in cuBLAS, using the "Rosetta stone" approach and the indexing methods that cuBLAS uses for converting 2D matrices into 1D vectors, etc.

And don't forget to fill in the values of 0, 0, 0, and 1, into the final row for matrix M!