Optimal Transformations between Matching Point Groups

Noah Litov

6/16/16

Summary

There are often real-world scenarios where one will have to find a point's inferred location relative to a set of points, and are only left with imperfect points correlating to the needed point group; this paper sets out to solve this problem. This question is frequent asked in relation to computer graphics as well as the science of metrology. In order to solve this problem, we must understand the different ways to represent a point's relationship to a point cloud, in numeric, vector, and matrix form. Additionally it is important to know how to relate point clouds to one another and find the relationship so we can transform the given point. Finally, and with the most difficulty, we need to find this relationship for an imperfect and differing set of points. We do this through either the Kabsch Algorithm of the Iterative Closest Point Algorithm.

Background Note: *In order to properly understand this paper, the reader is expected to have a working knowledge of trigonometry, vector geometry, and linear algebra. While not necessary, a basic understanding of integral calculus would be helpful. No prior information on this specific topic is needed.*

Introduction

This paper works to provide a solution for the problem of finding the location of a point in a $\mathbb{R}^3$coordinate system relative to a different, imperfect $\mathbb{R}^3$coordinate system. This solution involves defining terms for this solution, providing an understanding of point to point cloud relationships, exploring correlating point clouds, and finally finding a solution that will work with imperfect point groups.

Background

Many real-world situations deal with 3 dimensioned points in space. The inspiration for this research comes from the struggle to find point measurements relative to machinery using a Laser Tracker (a high-precision industrial measurement tool). This is difficult because the tool is only capable of measuring points relative to itself (it serves as the origin of the measurement and determines point location by spherical angles and a distance). In order to find these points relative to the machine, the "tracker's" relative location to the machine must be determined. This math also is very useful in computer graphics. It can be used in photo editing software to

match contours of graphics to one another, scale graphics, or transform the graphics on the screen.

In order to understand these point transformations, it is important to establish how to define these points mathematically. For the purpose of this research we can assume that these points are being defined in $\mathbb{R}^3$ (3 dimensional space). Basic math provides 3 common (fairly similar) ways to define points in $\mathbb{R}^3$:

- Numeric Coordinates: (x,y,z) or (r,θ,z) or (r.θ.ϕ)
- Matrices: $[x \quad y \quad z]$ or $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$
- Vector-Distances: <x,y,z> or $\{x\hat{\imath} + y\hat{\jmath} + z\hat{k}\}$.

Each of these different systems comes with its own set of math rules and provides a different medium to consider the problems at hand. The simplicity of the numeric system is usually what is defaulted to for simple transformations; however, as will be demonstrated in the following content, much of this math can be conducted more readily with matrices and vectors.

Section 1: Data Points to Point Clouds

It is important to determine an independent point's relationship to the others. When given a point in $\mathbb{R}^3$, the assumption is often made that the point is in regard to a given coordinate system with a known origin. However, if the coordinate system is not known, we must simply consider the point's relationship to all other known points. While occasionally the independent point will fit nicely into the geometry of the other points, often it would provide geometry too difficult to work with to be worthwhile considering. This leaves only the option of considering the relative distances between the independent point and all other points.

This can be done numerically. However, this is very difficult to do because there is no condensed way to describe these different relationships, so what is left is as many separate coordinate distances as there are points. It is also possible to consider the distance to the centroid of the point group, but that is a single distance relationship with no affiliated direction, and so could be anywhere on the surface of a sphere of possible locations around the independent point.

With vectors, the relationship is found similarly to with points; however, the relationship to the centroid is far more precise because, in addition to distance, we can now show *direction* from the point. The total relationship can also be expressed with a resultant vector-distance combining all the provided vectors. Assuming a shared coordinate system, this is the best way to represent the relationship between an independent point and a point cloud, because it is a very simple, precise, and visual relationship.

Matrices provide the most options of any of the systems. The distance relationships can be expressed as a set of vector distances represented by vertical column matrices. This in turn can be rewritten as a single matrix representing the set. The point cloud coordinates can be used to generate a basis that spans $\mathbb{R}^3$, map the independent point onto the new basis and have a precise relationship regardless of true origin.

We must also consider the number of points. Any of these methods should work regardless of the number of points given in the point cloud; however, if fewer than three points are used, the centroid of the cloud will provide a relationship to the independent point that resembles a ring normal to the distance between the points. This is because the independent point would have the same distances to the points no matter where on the ring it lays. With this in mind, it is safe to assume that using fewer than 3 points in a point cloud will lead to an imprecise relationship. With many points, the vector representation becomes far more difficult to find (the centroid relies upon a greater number of data points), and the numeric representation becomes almost impossible to find or demonstrate by hand. The matrix representation however, should remain almost identical since no matter the size of the point cloud, the basis should remain consistent, and the mapping will be equally easy.

Section 2: Correlating Point Clouds

It is also important to find point cloud's relationships to one another. A similar problem to what is presented in the first part of the research, relationships between point clouds can be thought of in two ways:

- The sum of the relationships between all correlating points
- The relationship of the point clouds as a mathematical entity.

Correlating point clouds simply implies that the geometry of the point cloud is sustained while the positions differ. This can mean that every point in a cloud correlates with a point in the other, or that there are sufficient correlating points between the clouds that geometry can be inferred, and the points that don't correlate are not defined in the opposing point cloud.

It is often helpful to consider real world situations to remove some of the abstraction from the concept. Consider a car. While standing behind the car, one can see the back edge of the roof, the left and right rear tires, the exhaust, the rear bumper and the license plate. Standing to the left of the car however, one can see the front and rear left tires, the back and front edges of the roof, the doors, and the front and rear bumper. In this situation, these points are in totally different relative spaces from one another, but the rear tire, bumper, and roof correlate between the point groups. Understanding the relationship between the correlating point groups means that one can infer the location of ANY of the points (even the non-correlating points) from either perspective, or another perspective entirely.

The best way to find this kind of relationship is by considering a transformation of the point group (this can be a translation or a rotation). When the point cloud is moved along this transformation, the correlating points should match each other. That transformation can then be used to describe the relationship between the groups more precisely than a distance can. Additionally, a transformation can be checked by shifting an unused correlating point and checking the distance between its related point in the new point group.

Numerically, this is once again a difficult concept since distance and direction cannot be described simultaneously. The way to find the relationship is either a highly abstract set of distances between correlating points, which doesn't allow much way to find an inferred point relative to a different point cloud. The other option is once again relating the centroids of ONLY the correlating points. However, similarly to relating a single point, this is a very abstract distance and cannot provide a direction, leaving an unacceptably flawed solution. This all but eliminates a numeric solution as a possibility.

Vector distance relationships are much closer, but still share some problems. When comparing centroids by vectors, a very accurate translational relation between the point groups can be generated; however it does not account for the rotational relation of the points (the centroids could line up, but the system needs to be rotated to match up). Likewise if individual correlating point relationships are found, every sequential transformation will undo whatever the last transformation accomplished, simply offering a series of transformations that do not relate to one another. The one other option for vectors *alone* is to consider the average transformation of all the distance vectors created between correlating points; this however, shares the same fatal flaw as the centroid transformation.

Finally, Linear Algebra has a well-established and studied method of handling these relationships known as a "Transformation Matrix." This is a concept and can be represented in several different ways, but the execution remains the same. When a vector representation of a point in subspace A is plugged into a transformation-matrix function from subspace A to subspace B, the output represents the position vector of that point in subspace B (the subspace from the second point cloud). This method allows for both translational and rotational relationships to be accounted for simultaneously. Finding the transformation matrix consists of finding the translational relationship and the rotational relationship between the different vector spaces and then plugging those into a set format of matrix.

Three major types of transformation matrices exist in standard schools of thought, these are ('A' being the vector transformed):

- Affine: $A\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & s \end{bmatrix}$, {A = $\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$} {t stands for translation, r for rotation, and s for scale}

- Linear: $A\begin{bmatrix} x \\ y \\ z \end{bmatrix}$, {A = $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$}
- Perspective: $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$, {A = $\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$}
- Rigid (Euclidean): $A\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} + \begin{bmatrix} x \\ y \\ z \end{bmatrix}$, {A = $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$}

Among these, only the Rigid Transform and the Affine Transform fit the situation. The linear transform only translates points, while the perspective transform only changes the scale of points. The rigid transform and the affine transformation can clearly be used interchangeably, so for the remainder of the paper, I will simply refer to "transformation matrix" when discussing either.

To utilize this relationship, we need to find the transformation matrix. The best method of doing this with directly relative points is to use diagonalization to find the eigenvalues and eigenvectors representing these point groups, and map between the eigenbases. While a very simple process in execution, this is a very abstract concept visually, and can be difficult to describe or picture.

Section 3: Imperfect Pairings

The next step is to figure out these relationships with imperfect groups. Up to this point, what has been discussed is ideal world situation. When all the "correlating" points have an exact match in the other point group. The challenge is when the points are not quite perfect. When the relationship between correlating points is not identical for each pair. This can commonly happen when relating real-world to conceptual world, and almost universally when comparing two sets of human-taken measurements.

Finding a transformation matrix gets much more difficult. The eigenvalue approach to finding a transformation matrix no longer works because with imperfect points the math will end up with a matrix that is not diagonalizable. Additionally, the mathematician must find some way to reduce these errors between points so the transformation leads to the smallest possible cumulative error. This added difficulty takes the problem from being a three step discovery to a many step difficult math problem.

Mathematicians today execute this process primarily with two algorithms, both are fairly recent discoveries. They are the:

- Iterative Closest Point (ICP) Algorithm
- Kabsch Algorithm.

Each algorithm is a series of steps that allow for finding the optimal transformation matrix, and both utilize a "reference cloud" where the focus is translating one point cloud to another fixed "reference" (assuming the reference to be the "perfect" cloud).

The ICP Algorithm takes a point by point approach to finding the transformation matrix, stepping through the following:

1. Find the closest point in final cloud to each point in the initial cloud
2. Find root mean squared error of the entire system
3. Estimate rotation and translation to decrease RMSE found above
4. Transform entire initial cloud by rotation and translation
5. Iterate steps 1-4 until RMSE cannot decrease
6. Combine all transformations from iteration to find net transformation

This is a long, brute-force process and can almost exclusively be done by computer processing hardware since there will often be hundreds of iterations before an optimal transformation is found. However, it can be done with any sets of points with any size error, and can leave the type of transformation matrix up to the user making it a useful tool.

The Kabsch Algorithm is a far more elegant solution to the problem. It follows the following steps once through and is complete:

1. Find the centroids of the point clouds
2. Find rotation
    a. Find the "Covariance Matrix"
        i. Translate both point groups to the origin by using their centroid
        ii. Multiply the initial set of points by the transpose of the final set
    b. Use "Singular Value Decomposition" to decompose the Covariance Matrix
        i. H (Covariance) = USV^T
    c. Multiply the transpose of U by V in order to get a 3x3 rotation matrix
        i. $R = U^TV$
3. Find translation = t
    a. Multiply the new rotation matrix by the centroid of the initial point group and then add the centroid of the final to find the difference between the two after rotation.
4. AR+t = New Rigid Transform

This algorithm is very simple and straightforward, and works for all sizes of point groups. Due to the long nature of Singular Value Decomposition, it also is better done with computer code, however all the other steps can easily be done by hand. It also factors in the error reduction with the Singular Value Decomposition, so when you check the RMSE of the newly translated points, you should end up almost universally with as small of a value as the provided point groups will allow.

Conclusion

This is a very difficult math problem that can boil down to a very simple, elegant solution, or a far less elegant, but practical solution. From our understanding of point representation, points' relationships to clouds, and clouds' relationships to one another, we are able to extract a concise algorithm that is applicable not only to perfect pairings but also, with consideration, imperfect clouds. The simplicity of the ending solutions means that anyone with a basic knowledge of Linear Algebra and Calculus can do the math, and anyone with some programming knowledge can find a way to implement it.