Increasing Robustness in Self-Localization and Pose Estimation

Ross J. Micheals Terrance E. Boult
VAST Lab, Lehigh University, Bethlehem, PA 18015
rmicheals@lehigh.edu, tboult@eecs.lehigh.edu

Abstract

Ideally, an algorithm used for either self localization or pose estimation would be both efficient and robust. Many researchers have based their techniques on the absolute orientation research of B. K. P. Horn. As will be shown in this paper, while Horn's method performs well with an additive Gaussian noise of large variance, mismatches and outliers have a more profound effect. In this paper, the authors develop a new closed-form solution to the absolute orientation problem, featuring techniques specifically designed for increasing the robustness during the critical rotation determination stage. We also include a comparative analysis of the various strengths and weaknesses of both Horn's and the new techniques.
Keywords*:* absolute orientation, self-localization, pose estimation

1 Introduction and Background

At the core of many pose estimations and self-localizations, lies a variation of the *absolute orientation* (AO) problem. Historically, and throughout different fields, the problem takes on both different constraints and names. But, regardless of whether the problem to be solved is "3D location parameter estimation" [16], a form of "hand-eye calibration", or (the authors' favorite nomenclature) the "roto-translation" problem [15], the essence remains the same. Given two sets of corresponding 3D points, what is the transform that relates them?

To the best of the authors' knowledge, the first published least-square minimization solution to the absolute orientation problem can be accredited to Fernando Sansò [15]. Horn rediscovered the same method over 14 years later, in 1987 [7]. Although Sansò's paper outdates Horn's, the latter is the significantly more popular reference. Both authors first reduce the problem to an Eigen-decomposition of the same 4×4 matrix, and then suggest the use of Jacobi's method. Unlike Sansò, Horn mentions that the Eigenvectors and Eigenvalues of the matrix may be found via the roots of a quartic. Since the roots of a quartic may be found in closed form via Ferrari's identities, Horn's completes his closed-form. Horn suggests, however, that due to the inherent instability of the closed-form solution for quartics, in implementation, the decomposition should be performed via Jacobi. (See [12] for more history on the AO problem.)

As defined in this paper, we consider the absolute orientation problem to be determining, from a collection of corresponding point pairs, the translational, rotational, and scalaring correspondence between two different Cartesian coordinate systems. In this paper, we will present a new closed-form quaternion based AO algorithm particularly well-suited for dealing with mismatches and outliers. Through simulation, we compare the new methods with previous research.

2 Representations

Choosing a representation for the scalar and translational components is simple — we use a scalar and a 3D vector respectively. Rotations, however, have many different representations (Horn reminds us both in [7] and [8] that Korn and Korn [11] mention five different methods alone). Aside from Euler angles, the rotation matrixes and quaternions are the representations most often used in the computer vision and graphics community (see [3] and [6] for background), but which of the many representations is best suited for the AO problem?

Like the research of Horn, we are primarily concerned with the quaternion representation of rotations. This bypasses the following disadvantages of using rotation matrixes:

- There is no obvious relationship between the individual elements of a rotation matrix and the axis & angle of rotation. This lack of an "intuitive" understanding makes both comparison and visualization difficult. Therefore, rotation matrixes are also poor candidates for reconciliation via averaging multiple results from subsampling noisy data.
- Traditional rotation matrixes and Euler angles make interpolation difficult. Computer vision, graphics, robotics, and manufacturing applications often have better results when using quaternions [10, 14].
- The normalization required to produce a numerically correct rotation matrix is computationally expensive. Orthogonalization procedures, such as Gram-Schmidt, introduce unnecessary opportunities for errors to propagate and can require significant amounts of computation.

Unit quaternions are well-suited for representing rotations for many situations:

+ Because of the direct relationship between the angle & axis of a rotation and the quaternion that represents it, groups of quaternions are easy to compare and reconcile.
+ Normalization of a quaternion is relatively fast, requiring significantly less work than an orthogonalization procedure.

Quaternions do have features that can be disadvantages:

± For every rotation, there exists two canonical quaternion representations; i.e. a rotation through $\mathring{q}$ is equivalent to

one through $-\mathring{q}$. Although this may appear to be a disadvantage, we will later exploit this dual nature of quaternions.

- Despite their correlation to the axis and angle of a rotation, the combination of a quaternion's four-dimensional nature with its unit constraint can be difficult to visualize. Quaternion visualization has a long history: from 1866 "desk and table" analogy [4] to Hart's [5] more recent (1994) use of computer graphics to explore the more common "belt" analogy.
- Quaternions give no preference to rotations that maintain an "upright" position. Therefore, in the domain of computer graphics, quaternions are not preferred for interpolating between virtual camera orientations [3].

3 Absolute Orientation Formulae

Consider a set of n three dimensional column vectors (perhaps representing points on a rigid body or geolocated landmarks) $R = \{\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_n\}$ where $\mathbf{r}_i^T = [r_{i_x}, r_{i_y}, r_{i_z}]$ and $1 \leq i \leq n$. Let S represent the corresponding set of n points after undergoing a translation by t, a uniform scaling by c, and a rotation by θ degrees about the unit axis $\hat{\mathbf{u}}$. If $M(\theta, \hat{\mathbf{u}})$ represents the matrix form of the rotational component as a function of θ and $\hat{\mathbf{u}}$, then for all $1 \leq i \leq n$,

$$\mathbf{s}_i = cM(\theta, \hat{\mathbf{u}})\mathbf{r}_i + \mathbf{t}. \tag{1}$$

Formally, the absolute orientation problem is: Given R and S, recover c, $\mathbf{t}$, θ, and $\hat{\mathbf{u}}$. Like previous research, we treat the absolute orientation in three separate stages: by solving c, $\mathbf{t}$, and (θ, $\hat{\mathbf{u}}$) independently. The focus of this research is on the most difficult component, determining the rotation. Therefore, we will only briefly summarize the common methods for finding the scalar and translational components.

3.1 Solving the Translational Component

To determine the translational component, $\mathbf{t}$, we look to the centroids. Given the respective centroids of each point set, $\mathbf{t}$ is simply the difference between the two vectors. Simply calculate the centroids

$$\mathbf{r}_c = \tfrac{1}{n}\textstyle\sum_{i=1}^{N} \mathbf{r}_i \quad \text{and} \quad \mathbf{s}_c = \tfrac{1}{n}\textstyle\sum_{i=1}^{N} \mathbf{s}_i \tag{2}$$

then the translational component $\mathbf{t}$ can be approximated by simply

$$\mathbf{t} = \mathbf{s}_c - \mathbf{r}_c. \tag{3}$$

3.2 Solving the Scalar Component

In the multi-stage approach, obtaining an estimate for the scalar component is also relatively trivial. Suppose that given a point set, we sum all of the distances from each point to the centroid of the set. If our point set remains rigid, we expect this total to be invariant under the effects of translation and rotation. Therefore, we look to the ratio of the average distance to the centroid for an estimate of c:

$$c = \left(\frac{\sum_{i=1}^{n} \|\mathbf{r} - \mathbf{r}_c\|}{\sum_{i=1}^{n} \|\mathbf{s} - \mathbf{s}_c\|} \right)^{\frac{1}{2}} \tag{4}$$

Horn [7] provides a more formal treatment of precisely why Equations (3) and (4) are desirable estimates in the absence of mismatches and outliers.

3.3 Solving the Rotational Component

Solving for the rotation is the crux of the new research. For the sake of simplicity, in this section we assume that our input data has *already been normalized* according to the scalar and translational components.

Consider a set of n three-dimensional column vectors (perhaps representing points on a rigid body) $R = \{ \mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N \}$ where $\mathbf{r}_k^T = [r_{k_x}, r_{k_y}, r_{k_z}]$ and $1 \leq k \leq N$. Let S represent the corresponding set of n points after undergoing a rotation of θ degrees around the unit axis $\hat{\mathbf{u}}$ (point $\mathbf{r}_k$ rotates to $\mathbf{s}_k$).

For convenience, vectors will be freely interchanged with their quaternion counterparts (i.e. $\mathring{r} = (0, \mathbf{r})$).

3.4 Outliers and Mismatches

A particular advantage of using the aforementioned methods is that neither require point matches. In other words, even if *all* of the correspondences are incorrect, then the translational and scalar components will not be effected. Determining the rotation, however, requires correspondences, even if they are implicit ones. The issues associated with this independence from correspondences will resurface when we consider the metrics to be used for evaluating AO algorithms.

In the presence of outliers, however, Equations (3) and (4) can rapidly deteriorate. It is the profound effect of outliers that motivated Fischler and Bolles to develop their landmark RANSAC (Random Sample Consensus) framework [2]. RANSAC instantiates not just one, but many solution models through the use of repeated subsampling. By seeking a majority of points consistent with their models, Fishler and Bolles were able to use RANSAC to increase the robustness of a classic photogrammetric localization problem. Later, our simulations will show how outliers may cause a similarly corrupting effect.

3.5 The New Closed-Form

For the sake of deriving the new form, we assume that our input data (R and S) is noiseless, complete, and correctly matched; in later sections we examine the effects of noise. If $\mathring{q}$ is the unit quaternion

$$\mathring{q} = (sin(\theta/2), cos(\theta/2)\hat{\mathbf{u}}) \tag{5}$$

it can be shown that the post-rotation vector, $\mathbf{s}_i$, is uniquely determined by the quaternion multiplication

$$(0, \mathbf{s}_i) = \mathring{q}(0, \mathbf{r}_i)\mathring{q}^* \tag{6}$$

where $(0, \mathbf{r}_i)$ is a quaternion with a scalar component of zero and vector component $\mathbf{r}_i$, and $\mathring{q}^*$ is the conjugate of quaternion $\mathring{q}$.

Fully expanding (s_{i_x}, s_{i_y}, and s_{i_z}) algebraically yields Equations (7)–(8) of Figure 1. From these equations we can see that the components of the sought-after quaternion can not be expressed as a direct linear combination of the input points.

However, if we consider the *product* of two of the quaternion components, we can establish a linear relationship.

Suppose we choose just three pairs of corresponding points from R and S: $(\mathbf{r}_1, \mathbf{s}_1)$, $(\mathbf{r}_2, \mathbf{s}_2)$, and $(\mathbf{r}_3, \mathbf{s}_3)$. We can then build the system of Equation (10) of Figure 1.

These ten equations fully constrain the rotation between R and S. It is no surprise that at least three points were needed. The inclusion of the unit quaternion identity $\|\mathring{q}\| = 1$ may not be as obvious, however. In fact, it was the inclusion of the unit quaternion constraint that allows **Maple**™, with some creative coaxing, to find closed-form solutions for all ten component combinations.

Although it is not the main focus of this paper, it should be noted that this system can easily be extended to include the determination of a translational component. The three additional degrees of freedom can be constrained with a fourth point pair. Despite its larger size, closed-form solutions for all 13 unknowns (10 quaternion products and 3 degrees of translation) can be found from the new system. Details are provided in [12].

After significant algebraic manipulation of **Maple**™'s output, we find that all products of two components of a (quaternion) rotation can be expressed as a linear combination of triple product ratios. In Equations (12)–(16), the component products are expressed as functions of the original input points.

The six component products $q_s q_x$, $q_s q_y$, $q_s q_z$, $q_x q_y$, $q_y q_z$, $q_x q_z$ are all of the form of Equation (16) where ξ and ζ are elements of $\{s, x, y, z\}$ and $\xi \neq \zeta$. For instance, if $\xi = s$ and $\zeta = x$, then the component considered is $q_s q_x$, substituting the matrix P_{sx} where appropriate. Note that $q_\xi q_\zeta = q_\zeta q_\xi$.

3.5.1 Computational Expense

A direct implementation of the above equations can hardly be considered an efficient one. Specifically, an actual implementation should not include any matrix multiplications — the matrixes of Equation (11) all reduce to element swaps and/or negations and have been included for notational compactness only. Common subexpression elimination also yields significant speedup.

As computing power continues its exponential growth along Moore's price-performance curve, the number of computations required by an algorithm is less important than in previous years. Even a typical desktop PC can run proverbial circles around the most powerful workstations of yesteryear. Horn's absolute orientation algorithm is a perfect example of how inexpensive cycles can improve algorithms. As disclosed in Section 1, even though the closed-form algorithm as described in [7] *can* be implemented in pure form, it usually is not. The high availability of mathematical libraries that rapidly perform iterative procedures have made it possible to substitute potentially unstable algorithms with more robust ones.

Still, there is no substitute for high-efficiency algorithms; embedded systems, robotics, RANSAC-based algorithms, and real-time systems all depend on rapid turn-around. Unless an algorithm is particularly compact, RANSAC algorithms especially can not afford to repeat an iterative numerical method hundreds or thousands of times.

Regardless, briefly consider a naive implementation of Equation (10). If we assume that a dot product requires three multiplications and one addition, a cross product requires six multiplications and three additions; then an implementation with no partial evaluation can find an estimate of the q_s^2 component with just 45 multiplications, 25 additions, and 1 division — just 71 FLOPS. A highly optimized version that solves for all of the quadratic components can be performed in 123 FLOPS and 37 assignments. (See [12] for implementation details.) An optimized method that calculates all ten components (which may not always be necessary) can be accomplished with 266 FLOPS.

3.6 Increasing Stability and Degenerate Cases

Clearly, as the volume of the parallelepiped spanned by $\mathbf{r}_1$, $\mathbf{r}_2$, and $\mathbf{r}_3$ approaches zero (all three points and the origin are coplanar), the solution degenerates. However, there is a simple way to increase the stability of the new form. If $\langle \mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3 \rangle$ is greater than $\langle \mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3 \rangle$, then use the formulations as written. However if $\langle \mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3 \rangle$ is greater, then the arguments to each component function should be $(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, \mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3)$ instead of $(\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3, \mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3)$. The conjugate of this result is the sought-after estimate.

While three point pairs are sufficient for the non-translation case, four pairs of non-coplanar points are required for both translation and rotation.

3.6.1 Resolving the Overdetermined System

Since not all ten components are *required* to calculate the sought-after quaternion, we are left with many possible reconstruction techniques. We can now use the dual nature of quaternions as an advantage; any one component can be arbitrarily fixed as positive, and the sign of other components can be adjusted accordingly. The resulting rotation is not effected. Since it has a direct correlation to the angle of rotation, it makes the most intuitive sense to fix the scalar component q_s positive.

During early experimentation, it was discovered that out of all the 10 products, q_s^2, q_x^2, q_y^2, and q_z^2 are the least susceptible to noise. Using the stable products as an anchor, and the less stable products for determining the proper sign, the rotation can be reconstructed with the algorithm shown in Figure 2.

In some cases, the size of the input set is minimal. In [9], only a handful of points (three to five) are used for localizing a mobile platform using geolocated landmarks. Using a greater number of points, however, has the potential for increasing many algorithms' accuracy.

Fortunately, there are many ways to generalize the absolute orientation problem to n points. For instance, one may seek the rotation that:

- minimizes the sum of the squared distances between the pre and post-rotated points
- maximizes the alignment of the data, treated as vectors, before and after the rotation
- minimizes the sum of squared errors in any linear formulation that produces a solution

$$s_{n_x} = r_x q_s^2 + r_x q_x^2 - r_x q_y^2 - r_x q_z^2 + 2r_y q_x q_y + 2r_z q_x q_z + 2r_z q_s q_y - 2r_y q_s q_z \tag{7}$$

$$s_{n_y} = r_y q_s^2 - r_y q_x^2 - r_y q_y^2 - r_y q_z^2 + 2r_x q_x q_y + 2r_y q_z q_z - 2r_z q_s q_x - 2r_x q_s q_z \tag{8}$$

$$s_{n_z} = r_z q_s^2 - r_z q_x^2 - r_z q_y^2 + r_x q_z^2 + 2r_y q_y q_z + 2r_x q_x q_z + 2r_y q_s q_x - 2r_x q_s q_y. \tag{9}$$

$$\begin{bmatrix}
0 & 2r_{1_z} & -2r_{1_y} & 2r_{1_y} & 0 & 2r_{1_z} & r_{1_x} & r_{1_x} & -r_{1_x} & -r_{1_x} \\
-2r_{1_z} & 0 & 2r_{1_x} & 2r_{1_x} & 2r_{1_z} & 0 & r_{1_y} & -r_{1_y} & r_{1_y} & -r_{1_y} \\
2r_{1_y} & -2r_{1_x} & 0 & 0 & 2r_{1_y} & 2r_{1_x} & r_{1_z} & -r_{1_z} & -r_{1_z} & -r_{1_z} \\
0 & 2r_{2_z} & -2r_{2_y} & 2r_{2_y} & 0 & 2r_{2_z} & r_{2_x} & r_{2_x} & -r_{2_x} & -r_{2_x} \\
-2r_{2_z} & 0 & 2r_{2_x} & 2r_{2_x} & 2r_{2_z} & 0 & r_{2_y} & -r_{2_y} & r_{2_y} & -r_{2_y} \\
2r_{2_y} & -2r_{2_x} & 0 & 0 & 2r_{2_y} & 2r_{2_x} & r_{2_z} & -r_{2_z} & -r_{2_z} & -r_{2_z} \\
0 & 2r_{3,z} & -2r_{3,y} & 2r_{3,y} & 0 & 2r_{3_z} & r_{3_x} & r_{3_x} & -r_{3_x} & -r_{3_x} \\
-2r_{3_z} & 0 & 2r_{3_x} & 2r_{3_x} & 2r_{3_z} & 0 & r_{3_y} & -r_{3_y} & r_{3_y} & -r_{3_y} \\
2r_{3_y} & -2r_{3_x} & 0 & 0 & 2r_{3_y} & 2r_{3_x} & r_{3_z} & -r_{3_z} & -r_{3_z} & -r_{3_z} \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1
\end{bmatrix}
\begin{bmatrix} q_s q_x \\ q_s q_y \\ q_s q_z \\ q_x q_y \\ q_y q_z \\ q_x q_z \\ q_s^2 \\ q_x^2 \\ q_y^2 \\ q_z^2 \end{bmatrix}
=
\begin{bmatrix} s_{1,x} \\ s_{1,y} \\ s_{1,z} \\ s_{2,x} \\ s_{2,y} \\ s_{2,z} \\ s_{3,x} \\ s_{3,y} \\ s_{3,z} \\ 1 \end{bmatrix} \tag{10}$$

$$P_{xx} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad P_{yy} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad P_{zz} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

$$P_{sx} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} \quad P_{sy} = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad P_{sz} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{11}$$

$$P_{xy} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad P_{yz} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad P_{xz} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

$$q_s^2 = \left| \frac{\langle \mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3 \rangle + \langle \mathbf{s}_1, \mathbf{r}_2, \mathbf{r}_3 \rangle + \langle \mathbf{r}_1, \mathbf{s}_2, \mathbf{r}_3 \rangle + \langle \mathbf{r}_1, \mathbf{r}_2, \mathbf{s}_3 \rangle}{4\langle \mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3 \rangle} \right| \tag{12}$$

$$q_x^2 = \left| \frac{\langle \mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3 \rangle - \langle P_{xx}\mathbf{s}_1, \mathbf{r}_2, \mathbf{r}_3 \rangle - \langle \mathbf{r}_1, P_{xx}\mathbf{s}_2, \mathbf{r}_3 \rangle - \langle \mathbf{r}_1, \mathbf{r}_2, P_{xx}\mathbf{s}_3 \rangle}{4\langle \mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3 \rangle} \right| \tag{13}$$

$$q_y^2 = \left| \frac{\langle \mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3 \rangle - \langle P_{yy}\mathbf{s}_1, \mathbf{r}_2, \mathbf{r}_3 \rangle - \langle \mathbf{r}_1, P_{yy}\mathbf{s}_2, \mathbf{r}_3 \rangle - \langle \mathbf{r}_1, \mathbf{r}_2, P_{yy}\mathbf{s}_3 \rangle}{4\langle \mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3 \rangle} \right| \tag{14}$$

$$q_z^2 = \left| \frac{\langle \mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3 \rangle - \langle P_{zz}\mathbf{s}_1, \mathbf{r}_2, \mathbf{r}_3 \rangle - \langle \mathbf{r}_1, P_{zz}\mathbf{s}_2, \mathbf{r}_3 \rangle - \langle \mathbf{r}_1, \mathbf{r}_2, P_{zz}\mathbf{s}_3 \rangle}{4\langle \mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3 \rangle} \right| \tag{15}$$

$$q_\xi q_\zeta = \frac{\langle P_{\xi\zeta}\mathbf{s}_1, \mathbf{r}_2, \mathbf{r}_3 \rangle + \langle \mathbf{r}_1, P_{\xi\zeta}\mathbf{s}_2, \mathbf{r}_3 \rangle + \langle \mathbf{r}_1, \mathbf{r}_2, P_{\xi\zeta}\mathbf{s}_3 \rangle}{4\langle \mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3 \rangle} \tag{16}$$

Figure 1. Equations 7–9: Fully expanded expressions for the elements of $\mathbf{r}_n$ after a quaternion rotation about $\mathring{q}$. **Equation 10**: The post-rotation points as a linear combination of products of quaternion components. **Equations 11–16**: Solutions to the system as ratios of triple products. Triple products are denoted with angle brackets.

Algorithm Basic Reconstruct-3
Input: Six vectors, two groups of three, each respectively representing points before $(\mathbf{r}, \mathbf{s}, \mathbf{t})$ and after $(\mathbf{r}', \mathbf{s}', \mathbf{t}')$ a rotation through unknown $\mathring{q}$.
Output: $\mathring{q}$

quaternion: $\mathring{q}$
real: $rst, rst', q_s, q_x, q_y, q_z, q_{sx}, q_{sy}, q_{sz}$
// Calculate the triple products
$rst \Leftarrow \langle \mathbf{r}, \mathbf{s}, \mathbf{t} \rangle$
$rst' \Leftarrow \langle \mathbf{r}', \mathbf{s}', \mathbf{t}' \rangle$
// For stability, use the larger triple product
if $(rst > rst')$ {
  $q_s \Leftarrow |q_s^2(\mathbf{r}, \mathbf{s}, \mathbf{t}, \mathbf{r}', \mathbf{s}', \mathbf{t}')|^{\frac{1}{2}}$
  $q_x \Leftarrow |q_x^2(\mathbf{r}, \mathbf{s}, \mathbf{t}, \mathbf{r}', \mathbf{s}', \mathbf{t}')|^{\frac{1}{2}}$
  $q_y \Leftarrow |q_y^2(\mathbf{r}, \mathbf{s}, \mathbf{t}, \mathbf{r}', \mathbf{s}', \mathbf{t}')|^{\frac{1}{2}}$
  $q_z \Leftarrow |q_z^2(\mathbf{r}, \mathbf{s}, \mathbf{t}, \mathbf{r}', \mathbf{s}', \mathbf{t}')|^{\frac{1}{2}}$
  $q_{sx} \Leftarrow q_{sx}(\mathbf{r}, \mathbf{s}, \mathbf{t}, \mathbf{r}', \mathbf{s}', \mathbf{t}')$
  $q_{sy} \Leftarrow q_{sy}(\mathbf{r}, \mathbf{s}, \mathbf{t}, \mathbf{r}', \mathbf{s}', \mathbf{t}')$
  $q_{sz} \Leftarrow q_{sz}(\mathbf{r}, \mathbf{s}, \mathbf{t}, \mathbf{r}', \mathbf{s}', \mathbf{t}')$
} else {
  $q_s \Leftarrow |q_s^2(\mathbf{r}', \mathbf{s}', \mathbf{t}', \mathbf{r}, \mathbf{s}, \mathbf{t})|^{\frac{1}{2}}$
  $q_x \Leftarrow |q_x^2(\mathbf{r}', \mathbf{s}', \mathbf{t}', \mathbf{r}, \mathbf{s}, \mathbf{t})|^{\frac{1}{2}}$
  $q_y \Leftarrow |q_y^2(\mathbf{r}', \mathbf{s}', \mathbf{t}', \mathbf{r}, \mathbf{s}, \mathbf{t})|^{\frac{1}{2}}$
  $q_z \Leftarrow |q_z^2(\mathbf{r}', \mathbf{s}', \mathbf{t}', \mathbf{r}, \mathbf{s}, \mathbf{t})|^{\frac{1}{2}}$
  $q_{sx} \Leftarrow q_{sx}(\mathbf{r}', \mathbf{s}', \mathbf{t}', \mathbf{r}, \mathbf{s}, \mathbf{t})$
  $q_{sy} \Leftarrow q_{sy}(\mathbf{r}', \mathbf{s}', \mathbf{t}', \mathbf{r}, \mathbf{s}, \mathbf{t})$
  $q_{sz} \Leftarrow q_{sz}(\mathbf{r}', \mathbf{s}', \mathbf{t}', \mathbf{r}, \mathbf{s}, \mathbf{t})$
}
// Perform sign correction if necessary
if $(q_{sx} < 0)\ q_x \Leftarrow -q_x$
if $(q_{sy} < 0)\ q_y \Leftarrow -q_y$
if $(q_{sz} < 0)\ q_z \Leftarrow -q_z$
if $(rst' > rst)\ \mathring{q} \Leftarrow \mathring{q}^*$
$\mathring{q} \Leftarrow \mathring{q}/\|\mathring{q}\|$
return $\mathring{q}$
}

Figure 2. Basic Reconstruct-3 Pseudo-code. Given three pairs of input points, find the inverse rotation. The "⇐" symbol has been used to denote assignment.

- minimizes the median distance between the pre and post-rotated points

Obviously, there exist many other possible criteria that can obtain a desirable absolute orientation. They are likely to change based on the domain of the application. Least-squares approaches are often considered, but they presume only a Gaussian noise model, and therefore (implicitly) that there are no outliers. (For a powerful demonstration of the effect of a single outlier see [2]). In addition, they are often computationally intensive. It would be possible to solve Equation (10) in a least-squares sense via SVD, but we would not be freed from the side-effects of outliers. A RANSAC-like approach has the ability to combat many of these side-effects.

4 Absolute Orientation Algorithms

Now that the basic formulae have been introduced, we use them as a base for developing new algorithms. Since in Horn's solution there is a direct map from his formulation to an algorithm, we refer the reader to Horn's original paper for both reference and implementation [7].

The algorithms developed by the authors in this section have been developed in an informal and experimental fashion; i.e. they are not founded in a new fundamental theory. In this thesis, there are no proofs about the effectiveness of the new algorithm. However, we will show results from our simulations in which the new algorithm clearly outperforms previous research.

4.1 Quantifying the Effects of Noise

Early in the stages of this research, it became clear if the effects of noise and outliers could be quantized, then it would be easier to identify desireable estimates. Because a noise measure may need to be calculated for several thousand estimates, efficiency becomes a vital constraint of this process.

Let $\mathring{q}' = (q'_s, q'_x, q'_y, q'_z)$ represent the calculated value of $\mathring{q}$ from an algorithm such as *Reconstruct-N*. Let q'_{s^2} represent $q'_s q'_s$, q'_{sx} represent $q'_s q'_x$, and so on. If both our pre-rotation or post-rotation point sets are noiseless, then with the exception of floating point rounding errors, if our points are matched correctly, we can expect the following equalities to remain true:

$$q'_{s^2} q'_{x^2} = (q'_{sx})^2 \quad q'_{s^2} q'_{y^2} = (q'_{sy})^2 \tag{17}$$

$$q'_{s^2} q'_{z^2} = (q'_{sz})^2 \quad q'_{x^2} q'_{y^2} = (q'_{xy})^2 \tag{18}$$

$$q'_{y^2} q'_{z^2} = (q'_{yz})^2 \quad q'_{x^2} q'_{z^2} = (q'_{xz})^2 \tag{19}$$

In the presence of noise, one could not expect these identities to remain valid. However, it can be useful to look at the sum of the differences between the ideals of the identities and those values which are calculated. Let p (for *performance*) represent this difference

$$\begin{aligned} p = \; & |q'_{s^2} q'_{x^2} - (q'_{sx})^2| + |q'_{s^2} q'_{y^2} - (q'_{sy})^2| + \\ & |q'_{s^2} q'_{z^2} - (q'_{sz})^2| + |q'_{x^2} q'_{y^2} - (q'_{xy})^2| + \\ & |q'_{y^2} q'_{z^2} - (q'_{yz})^2| + |q'_{x^2} q'_{z^2} - (q'_{xz})^2|. \end{aligned}$$

As will be shown by the results of our simulation, in the general case, there exists an inherent correlation between quaternions with a low p value and the original rotation. In our previous algorithms, the goal was merely to generate a collection of similar quaternions. A simple way to exploit our newly found quantization ability would be to use p to build a *weighted sum* for each component.

4.2 Micheals-Boult

The first step in developing such an algorithm is to rewrite *Basic Reconstruct-3* to include the calculation of the performance metric. Figure 3 shows a pseudo-code implementation of this new variation. It may appear that calculating p could be a significant addition to our computational overhead, but the reader should be reminded that an optimized technique can be used. Regardless, our payoff is an ability to rate each estimate.

Algorithm Scored Reconstruct-3

Input: Six vectors — two groups of three respectively representing points before, $(\mathbf{r}, \mathbf{s}, \mathbf{t})$, and after, $(\mathbf{r}', \mathbf{s}', \mathbf{t}')$, a rotation through unknown $\mathring{q}$.

Output: $\mathring{q}$, p (or *performance*)

quaternion: $\mathring{q}$
real: rst, rst', q_s, q_x, q_y, q_z, $q_s x$, $q_s y$, $q_s z$, p
// Calculate the triple products
$rst \Leftarrow \langle \mathbf{r}, \mathbf{s}, \mathbf{t} \rangle$
$rst' \Leftarrow \langle \mathbf{r}', \mathbf{s}', \mathbf{t}' \rangle$
// Use the larger triple product for stability
if $(rst > rst')$ {
  $q_s \Leftarrow |q_s^2(\mathbf{r}, \mathbf{s}, \mathbf{t}, \mathbf{r}', \mathbf{s}', \mathbf{t}')|^{\frac{1}{2}}$
  $q_x \Leftarrow |q_x^2(\mathbf{r}, \mathbf{s}, \mathbf{t}, \mathbf{r}', \mathbf{s}', \mathbf{t}')|^{\frac{1}{2}}$
  $q_y \Leftarrow |q_y^2(\mathbf{r}, \mathbf{s}, \mathbf{t}, \mathbf{r}', \mathbf{s}', \mathbf{t}')|^{\frac{1}{2}}$
  $q_z \Leftarrow |q_z^2(\mathbf{r}, \mathbf{s}, \mathbf{t}, \mathbf{r}', \mathbf{s}', \mathbf{t}')|^{\frac{1}{2}}$
  $q_{sx} \Leftarrow q_{sx}(\mathbf{r}, \mathbf{s}, \mathbf{t}, \mathbf{r}', \mathbf{s}', \mathbf{t}')$
  $q_{sy} \Leftarrow q_{sy}(\mathbf{r}, \mathbf{s}, \mathbf{t}, \mathbf{r}', \mathbf{s}', \mathbf{t}')$
  $q_{sz} \Leftarrow q_{sz}(\mathbf{r}, \mathbf{s}, \mathbf{t}, \mathbf{r}', \mathbf{s}', \mathbf{t}')$
} else {
  $q_s \Leftarrow |q_s^2(\mathbf{r}', \mathbf{s}', \mathbf{t}', \mathbf{r}, \mathbf{s}, \mathbf{t})|^{\frac{1}{2}}$
  $q_x \Leftarrow |q_x^2(\mathbf{r}', \mathbf{s}', \mathbf{t}', \mathbf{r}, \mathbf{s}, \mathbf{t})|^{\frac{1}{2}}$
  $q_y \Leftarrow |q_y^2(\mathbf{r}', \mathbf{s}', \mathbf{t}', \mathbf{r}, \mathbf{s}, \mathbf{t})|^{\frac{1}{2}}$
  $q_z \Leftarrow |q_z^2(\mathbf{r}', \mathbf{s}', \mathbf{t}', \mathbf{r}, \mathbf{s}, \mathbf{t})|^{\frac{1}{2}}$
  $q_{sx} \Leftarrow q_{sx}(\mathbf{r}', \mathbf{s}', \mathbf{t}', \mathbf{r}, \mathbf{s}, \mathbf{t})$
  $q_{sy} \Leftarrow q_{sy}(\mathbf{r}', \mathbf{s}', \mathbf{t}', \mathbf{r}, \mathbf{s}, \mathbf{t})$
  $q_{sz} \Leftarrow q_{sz}(\mathbf{r}', \mathbf{s}', \mathbf{t}', \mathbf{r}, \mathbf{s}, \mathbf{t})$
}
// Calculate the estimate's performance
$p \Leftarrow |q_s^2 q_x^2 - q_s x^2| + |q_s^2 q_y^2 - q_s y^2| + |q_s^2 q_z^2 - q_s z^2| + |q_x^2 q_y^2 - q_x y^2| + |q_y^2 q_z^2 - q_y z^2| + |q_x^2 q_z^2 - q_x z^2|$
// Perform sign correction if necessary
if $(q_{sx} < 0)$ $q_x \Leftarrow -q_x$
if $(q_{sy} < 0)$ $q_y \Leftarrow -q_y$
if $(q_{sz} < 0)$ $q_z \Leftarrow -q_z$
if $(rst' > rst)$ $\mathring{q} \Leftarrow \mathring{q}^*$
$\mathring{q} \Leftarrow \mathring{q}/\|\mathring{q}\|$
return $\{\mathring{q}, p\}$
}

Figure 3. Weighted Reconstruct-3 Pseudo-code. Given three pairs of input points, find the inverse rotation and its corresponding performance metric p.

One last step remains; we must choose a weighting function. Obviously, as p decreases, we require a greater weight. In simulation, *Micheals-Boult* (Figure 4) weights each quaternion component by $1/p^2$. This weighting function has the desirable qualities of rapidly increasing as p becomes very small, almost negligible, for large values for p. Additionally, early experimentation suggested that $1/p^2$ was well-suited for the typically small (less than 1.0) magnitudes of the estimates. The weighting functions of $1/p$ and $1/p^3$ were also considered in early algorithms, but they did not produce more accurate results than $1/p^2$. For speedup, a piecewise linear approximation could be used.

Although lower values of p do not always correspond to desirable estimates, it is a measure that is independent of the general magnitudes of both the input and the noise vectors.[1]

Algorithm Micheals-Boult

Input: A set of n (where $n \geq 4$) vectors representing points before R, and after, R' undergoing an unknown translation **t** and an unknown rotation through $\mathring{q}$.

Output: $\mathring{q}$ and **t**

vector: **c**, **c**′, **r**, **r**′, **s**, **s**′, **t**, **t**′
quaternion: $\mathring{q}$
real: *WeightedQ*$_s$*Sum*, *WeightedQ*$_x$*Sum*
real: *WeightedQ*$_y$*Sum*, *WeightedQ*$_z$*Sum*
real: *SumOfWeights*
real: *score*, p
// Calculate the centroids for normalization and the
// determination of the translational component.
c $\Leftarrow$ *centroid*(R)
c′ $\Leftarrow$ *centroid*(R')
// For each contiguous group of three points
for $i = 1$ **to** $(n-2)$ **do**
  r $\Leftarrow R_i - \mathbf{c}$
  r′ $\Leftarrow R'_i - \mathbf{c}'$
  s $\Leftarrow R_{i+1} - \mathbf{c}$
  s′ $\Leftarrow R'_{i+1} - \mathbf{c}'$
  t $\Leftarrow R_{i+2} - \mathbf{c}$
  t′ $\Leftarrow R'_{i+2} - \mathbf{c}'$
  // Calculate an estimate and its performance
  $(\mathring{q}, p) \Leftarrow$ **Scored Reconstruct-3**$(\mathbf{r}, \mathbf{s}, \mathbf{t}, \mathbf{r}', \mathbf{s}', \mathbf{t}')$
  // Weight the performance
  score $\Leftarrow 1/p^2$
  // Contribute to each component according to the *score*
  WeightedQ$_s$*Sum* $\Leftarrow q_s \cdot$ *score*
  WeightedQ$_x$*Sum* $\Leftarrow q_x \cdot$ *score*
  WeightedQ$_y$*Sum* $\Leftarrow q_y \cdot$ *score*
  WeightedQ$_z$*Sum* $\Leftarrow q_z \cdot$ *score*
  SumOfWeights $\Leftarrow$ *SumOfWeights* + *score*
end for
// Extract each component
$q_s \Leftarrow$ *WeightedQ*$_s$*Sum* / *SumOfWeights*
$q_x \Leftarrow$ *WeightedQ*$_x$*Sum* / *SumOfWeights*
$q_y \Leftarrow$ *WeightedQ*$_y$*Sum* / *SumOfWeights*
$q_z \Leftarrow$ *WeightedQ*$_z$*Sum* / *SumOfWeights*
// Normalize the final model
$\mathring{q} \Leftarrow \mathring{q}/\|\mathring{q}\|$
// Calculate the translational component
t $\Leftarrow$ **c**′ $-$ **c**
return $\{\mathring{q}, \mathbf{t}\}$

Figure 4. Micheals-Boult pseudo-code. Using the performance metric, build weighted sums for each component from each quaternion estimate.

4.2.1 On the Merits of Micheals-Boult

The algorithm of Figure 4, *Micheals-Boult*, has specific provisions for handling outliers and mismatches while solving

[1] Be careful of our somewhat sloppy use of the term "magnitude" here. Certainly, as the magnitude of the typical noise vector increases, estimates will degenerate. But, if the ratio of the average noise vector magnitude to the average input vector remains the same, then we do not expect p to change dramatically. A distance measure would scale accordingly.

the rotational component. Determining precisely which is the most robust method for reconstructing the scalar and translational component is a part of the authors' continuing research. There are two methods currently under consideration:

1. Use the closed-form solutions to the extended matrix mentioned in Section 3.5 to build a translation capable version of *Micheals-Boult*.
2. Use a simple varation on RANSAC.

The authors' continuing research includes the determination of which method provides better results.

In addition, experimentation by the authors also suggests that the results from *Micheals-Boult* may be significantly improved by gathering estimates from more than $n-3$ samples. As seen in Figure 4, the new algorithm produces estimates through a simple sliding window. More intelligent subsampling techniques can significantly improve results.

5 Simulations

To evaluate the efficacy of the new algorithms, we developed a small-scale, but full-featured simulation system. In this section we will first discuss the simulation software used to test and evaluate the performance of both the existing and the new absolute orientation algorithms. Then, we present the results of the simulations.

The simulation software was written in an object-oriented nature with C++. All of the random number generators used in the simulation software are based on the source code provided in [13]. Robert Davies' *newmat09* library [1] was used for many of the matrix applications needed for the implementation of Horn's method.

5.1 Overall Process

The overall process of the simulation is straightforward. Generate two collections of vectors — one to represent points before a rotation and translation, and the other to represent the corresponding points after. We choose to disregard the scalar component in our simulations. The goal of the simulation is to use Horn's methods and the new algorithm to re-extract the transform.

In outlined form, the simulation performs the following:

1. Generate a set of pre-rotation points R.
2. Generate a random rotation $\mathring{q}$.
3. Using the pre-rotation points R and the rotation $\mathring{q}$, generate a corresponding set of post-rotation points S.
4. Generate outliers and mismatches.
5. Run an absolute orientation algorithm.
6. Using a pre-selected set of specific metrics, evaluate the performance of the algorithm.

Our simulation differentiates from previous research by including more realistic noise phenomena. In addition to the additive Gaussian noise, our simulation includes explicit steps to include various types of outliers.

As an aside, in the context of our simulation, when we refer to a "set" it is not in the mathematical sense, but more like an array. This allows us to use one index to refer to both the pre and post-transform point without an additional data structure. Using a single index also becomes handy later, when we generate synthetic mismatches.

Some of the important parameters in our simulation are the following:

m_p = mismatch probability
ω_p = outlier probability
Ω = outlier magnitude
n = number of points
ρ = radius of sphere
T = maximum distance of t to the origin
σ = magnitude of additive Gaussian noise

The specific role of each parameter will be explained as it comes into context. We will now examine each major step of the simulation in greater detail.

5.1.1 Generate Pre-Transform Points

First, we create the pre-transform data set. The simulation generates a collection of n random vectors R that could represent points on the surface of a bumpy sphere of radius ρ, and an origin at t. Noise is simulated with a simple additive Gaussian model, with an equal variance σ^2 in the axial directions. Random vectors from the model are added to the original points to create the noisy pre-rotation data set R_n.

5.1.2 Generate Rotation

First, to generate a random rotation we choose four random numbers from a uniform distribution from -1.0 to 1.0. Each of the random numbers is then assigned to an individual component of a quaternion $\mathring{q}$. For the sake of convenience, the scalar component of $\mathring{q}$ is fixed as positive, with an appropriate adjustment to the vector component. Finally, $\mathring{q}$ is normalized.

5.2 Generate Post-Transform Points

Using the pre-rotation vectors R and the rotation $\mathring{q}$, first generate their noiseless counterparts, S, adding in the random translation $\mathbf{t}'$ to each point as we go. Next, an additional Gaussian noise vector, also of variance σ^2, is added to each element in S to create the set of noisy post-rotation vectors S'.

It is at this point that the simulation significantly departs from previous research, by modeling two types of real-life error phenomena. First, it is possible that our correspondences themselves are importect, i.e. they include a number of *mismatches*. Second, an AO algorithm may accidentally include *outliers* — which we categorize as extremely noisy points. For example, a bad stereo correspondence or a human error could both generate an outlier.

5.2.1 Simulating Outliers

To simulate the presence of outliers, we start by copying the noisy sets of R_n and S_n into new outlier sets R_ω and S_ω. Then, for each point $\mathbf{p}_i$ in R_ω and S_ω, we choose a random floating-point number k from a uniform distribution spanning 0.0 to 1.0. If k is less than the input parameter ω_p, then we generate a point of random direction and random magnitude (of at maximum, Ω,) and replace $\mathbf{p}_i$ with the outlier.

5.2.2 Simulating Mismatches

The partially matched set, S_m, is generated in a similar manner. Due to the symmetry of mismatches, there is no need to generate a pre-rotation mismatch set R_m.

The mismatch set, S_m, starts as a duplicate of S_ω. For each point $\mathbf{s}_i$ in S_n, we choose a random floating-point number k', from a uniform distribution spanning 0.0 to 1.0. If k' is less than the input parameter m_p, then we choose a point at random from S_n, and replace $\mathbf{s}_i$ with its value. No special consideration is taken if $\mathbf{s}_i$ is already an outlier. After its generation, S_m contains a partially matched post-rotation set with outliers. Therefore, a point in R_ω, and its corresponding point in S_m, could be any of the following types of pairs:

- a correctly matched noisy pre-transform point paired with its corresponding noisy post-transform point
- a pre-transform outlier paired with a noisy post-transform point
- a noisy pre-transform point paired with a post-transform outlier
- a pre-transform outlier paired with a post-transform outlier
- a pre-transform noisy point incorrectly matched with a noisy post-transform point
- and so on ...

In summary, after this stage, we have generated

R_n = noisy set of pre-rotation points
R_ω = noisy set of pre-rotation points, including outliers
$\mathring{q}$ = rotation used to generate post-rotation points
S_n = noisy set of post-rotation points
S_ω = noisy set of post-rotation points, including outliers
S_m = noisy set of partially matched post-rotation points, including outliers)

and are ready to run the algorithms now.

5.2.3 Running the Algorithms

The two algorithms to be analyzed are Horn's method, and Micheals-Boult, which we will refer to as *Horn* and *Micheals-Boult* (or *M-B*). Because we have several different types of pre and post-rotation point sets, we must choose which sets are most appropriate for the different parts of the algorithms. There are essentially two stages to the simulation:

1. Determine the translational component, **t**
2. Determine the rotational component, $\mathring{q}$

For the translational component, we will calculate the centroids from R_ω and S_ω — this is the data an algorithm typically has access to. When outliers are present, we can expect the centroids (and thus the rotational components) to change dramatically. Since matches are inherently a component of determining the rotation, it does not make sense to use R_ω and S_m here. Instead, we reserve the use of S_m for the determination of the rotational component.

As mentioned in Section 3.5.1, Horn's method is rarely implemented in closed form. Therefore, our simulations do not implement *Horn* is closed-form.

5.3 Metrics

A standard part of evaluating any algorithm is determining the metrics to be used to quantify their performance. In this section, we will select our metrics, discuss how they should be used in our simulation, and the significance of their application. Although these metrics have been developed in the light of *Horn* and *Micheals-Boult*, they are general enough to apply to other AO methods.

5.3.1 Absolute Quaternion Distance (AQD)

A metric (to the best of the authors' knowledge) that has not yet been used to evaluate absolute orientation algorithms is a measure we will refer to as the **absolute quaternion distance**, or AQD. Essentially, the AQD is a measure of proximity between the quaternion used to generate the post rotation points $\mathring{q}$ and the quaternion estimated from an algorithm.[2] Given $\mathring{q}$ and $\mathring{q}'$, the AQD is trivial to compute:

$$\mathrm{AQD}(\mathring{q}, \mathring{q}') = \|\mathring{q} - \mathring{q}'\| \tag{20}$$

The AQD is not significantly perturbed in the presence of outliers, although precisely how an AO *algorithm* responds to outliers is entirely different. Just a single outlier can have a significant effect on the ADM, perhaps turning what may be an accurate estimate into an invalid one.

5.3.2 Average Distance Metric (ADM)

Traditionally, the goal of AO algorithms is to find the rotation that minimizes the average distance between the rotated pre-rotation points R' and the post-rotation points of S'. We define the **average distance metric**, or ADM, as a function of: two point sets (R and S), a pair of centroids ($\mathbf{r}_c$ and $\mathbf{s}_c$), and a rotation ($\mathring{q}$):

$$\mathrm{ADM}(R, S, \mathbf{r}_c, \mathbf{s}_c, \mathring{q}) = \sum_{i=1}^{n} \|(\mathbf{s}_i - \mathbf{s}_c) - \mathring{q}(0, \mathbf{r}_i - \mathbf{r}_c)\mathring{q}^*\|. \tag{21}$$

Consider a more general form of the ADM — one that is a function of *four* data sets, (R, S, R_c, and S_c) and a rotation ($\mathring{q}$), where R and S represent the pre and post-transform data sets, and R_c and S_c represent the data sets the algorithm uses to generate the centroids. If $\gamma(X)$ is a function returning the centroid of point set X, Then $\mathrm{ADM}(R, S, R_c, S_c, \mathring{q})$ equals

$$\sum_{i=1}^{n} \|(\mathbf{s}'_i - \gamma(S_c)) - \mathring{q}(0, \mathbf{s}_i - \gamma(R_c))\mathring{q}^*\| \tag{22}$$

In our simulation, outliers and mismatches are treated as separate phenomena. Therefore, this more general form of the ADM, which we will revisit shortly, provides us with a method of evaluating different variations of the ADM.

Since our simulation includes both outliers and mismatches, the application of the various metrics on the different pre and post-transformation data sets take on significantly different meaning. Therefore, we need to discuss the precise metrics that we will apply to our results.

[2] Obviously, the AQD is only appropriate in the presence of ground truth.

Ground Truth ADM The first metric of interest is the most traditional one – a standard application of the ADM, which we will refer to as the **ground truth ADM** or ADM-GT. In the new notation

$$\text{ADM-GT}(\mathring{q}) = \text{ADM}(R_n, S_n, R_n, S_n, \mathring{q}) \quad (23)$$

Note that the point sets used to generate the rotation are the same ones used to generate the centroids. To be true to tradition, the ADM-GT explicitly does *not* include outliers, implicitly does not include mismatches (otherwise the second parameter would be S_m and not S_ω) and is built only from the basic noisy point sets. Any method (including *Horn*) that finds the rotation that minimizes the sum of the squared distances between the pre and post-transformation points will share the same, minimum, ADM-GT. Therefore, we expect that when the input is correctly matched and lacks outliers is theoretically impossible to outperform Horn on the ADM-GT metric. As we will see in Section 6, on rare occasions *Micheals-Boult* does achieve lower values for ADM-GT than Horn. This is most likely due to numerical instabilities within Jacobian Eigen-decomposition, very small quaternion rotations, and typical floating-point roundoff errors.

Experimental ADM The next metric we will consider is modeled after the metric commonly used in an actual experimention. The **experimental ADM** or ADM-E is defined as

$$\text{ADM-E}(\mathring{q}) = \text{ADM}(R_\omega, S_m, R_\omega, S_\omega, \mathring{q}) \quad (24)$$

While this metric may seem straightforward, there are significant issues to consider when using it as an algorithm benchmark. First, even in the presence of a single outlier, the ADM-E may become very large, and give the impression that an estimate is less accurate than it actually is. Similarly, we can expect that a mismatch, a special kind of outlier, would have a similar effect on the ADM-E.

The presence of mismatches significantly changes the very nature of the AO problem. In *Horn*, a fundamental assumption is that all points are correctly matched. In this simulation, and in actual experimentation, however, one can not always make that assumption. Therefore, because the original assumptions made by Horn (and methods like his) are no longer valid, it is possible to outscore existing algorithms on both the ADM-GT and the ADM-E.

Cleaned ADM In our final ADM-based metric, we take advantage of our knowledge of ground truth. The **cleaned ADM**, or ADM-C, is generated in the same method as the ADM-E, except that points known to be either mismatches or outliers, are explicitly ignored. Let R_c (c for *cleaned*) and S_c represent

$$\begin{aligned} R_c &= \{\mathbf{r}_{n_i} \in R_n : \\ &\quad (\mathbf{r}_{n_i} = \mathbf{r}_{\omega_i}) \wedge (\mathbf{s}_{n_i} = \mathbf{s}_{\omega_i}) \wedge (\mathbf{s}_{n_i} = \mathbf{s}_{m_i})\} \quad (25) \\ S_c &= \{\mathbf{s}_{n_i} \in S_n : \\ &\quad (\mathbf{r}_{n_i} = \mathbf{r}_{\omega_i}) \wedge (\mathbf{s}_{n_i} = \mathbf{s}_{\omega_i}) \wedge (\mathbf{s}_{n_i} = \mathbf{s}_{m_i})\}. \quad (26) \end{aligned}$$

Essentially, R_c and S_c are the sets of correctly matched, non-outlier, noisy point pairs that survived the point generation process. Therefore, the ADM-C, or

$$\text{ADM-C}(\mathring{q}) = \text{ADM}(R_c, S_c, R_c, S_c, \mathring{q}), \quad (27)$$

is a reflection of the estimates' performance with respect to valid data, despite the presence of outliers and mismatches. In other words, the ADM-C is calculated from a cleaned subset of the points used to calculate the ADM-GT.

6 Results

Figures 5–16 show typical results from our simulations in the form of normalized histogram images.

For each iteration of the simulation, a single point was plotted on a graph with the point's x coordinate corresponding to Horn's value for a particular metric and the point's y coordinate corresponding to the Micheals-Boult value for that same metric. In these graphs, a point above the main diagonal indicates an iteration in which the Micheals-Boult value for the metric was *lower* than Horn's value. A point that falls below the diagonal corresponds to the inverse. Then, the plot is quantized into pixel regions and pixel values are assigned according to the number of values in each pixel region. Regions with the highest point count are assigned the darkest values and regions with the lowest point counts (usually zero) are assigned the lightest values. Figures 5–16 were all made from 10^5 iterations.

Figures 5 and 6 show the results from the simulation when its parameters are adjusted to mimic the data sets considered in traditional AO research. In this configuration, $m_p = 0$ (there are no mismatches), $p \approx 5.0$, $\omega_p = 0.0$ (there are no outliers), $n = 20$, $T = 10.0$ and we have a moderate noise of $\sigma = 0.2$ (approximately 4% of the radius of our bumpy sphere).[3] As expected, Horn's method consistently outperforms Micheals-Boult on both the ADM-GT and the AQD. The high density regions of Figure 5 indicate that even though over 97% of the points are in Horn's favor, in a large number of iterations, Micheals-Boult and Horn produced comparable results. In Figure 6, the vertical nature of the densest regions indicate that the AQD values from *Horn* remained within a much tighter and lower range than the AQD values from *Micheals-Boult*.

In Figures 7 through 10 we show the results from a set of parameters in which *Micheals-Boult* can perform significantly better than *Horn*. In our new configuration, we have raised the probability of mismatches, m_p, to 0.30, but lowered the noise, σ to 0.05, or approximately 1% of the bumpy sphere's radius. The other parameters remain the same: $\omega_p = 0.0$, $\rho \approx 5$, $n = 20$, and $T = 10$. As can be seen in the ADM-GT (Figure 8), ADM-C (Figure 9), and AQD (Figure 10) graphs, most of the data points fall significantly below the main diagonal in a linear-like cluster. This reflects our findings when performing simulations with similar parameters; that with low noise and moderate probability of mismatch, the majority of the Micheals-Boult metric values tend to remain within the same interval, while Horn's tend to have a much greater spread. In comparison to the ADM-GT, the lower density regions of

[3] Since there are neither mismatches nor outliers, both the ADM-E and ADM-C graphs are the same as Figure 5

ADM-C cover a smaller region. This reflects our intuitive understanding of the difference between the ADM-GT and the ADM-E — some of the points used in calculating the ADM-GT are never considered in either *Horn* or *Micheals-Boult*.

The graph of the ADM-E is quite different from its ADM counterparts. In Figure 7, the majority of the results are below the main diagonal (in favor of *Micheals-Boult*), but the orientation of cluster is along a diagonal of a slightly lower slope. If just ADM-E is considered, then one can legitimately draw the conclusion that in this configuration, *Micheals-Boult* and Horn produce comparable estimates. However, the high-density regions of ADM-GT and ADM-C, clearly show that *Micheals-Boult* posesses higher resiliency to mismatches than *Horn*.

Figures 11 through 16 show the results of two different variations of the parameters used from Figures 8–10. In both varations, mismatches have been eliminated ($m_p = 0.0$), but the outlier probability of maximum magnitude $\Omega = 20.0$, has been increased from $\omega_p = 0.0$ to $\omega_p = 0.10$. The other (non-translation) parameters have remained the same: $\rho \approx 5$, $\sigma = 0.05$, and $n = 20$.

In the first variation, we assume that the data sets have a known translational component, **t**. In other words, outliers only have effect in the determination of the rotational component. In the second variation, we do not assume a known rotation ($T = 10$); therefore both the translational and rotational components are calculated from the outlier-laden data.

Figures 11–13 show the results from the variation where **t** is known. The density distributions of these figures are quite similar to Figures 5–6. No doubt due to the magnitude of the outliers ($\Omega = 20.0$), the data points span a broader range. Again, we see that the ADM-E (Figure 12) shows that the application of an ADM to uncorrected data may give the impression that there is a nominal difference between *Horn* and *Micheals-Boult*.

Figures 14–16 show the results from the variation where **t** is unknown. Clearly, from the nearly $50/50\%$ split across all three metrics shown, without a more robust translation determination stage, neither algorithm performs particularly well. The ADM-E (Figure 15) continues its deceptive trend, but is skewed in favor of neither algorithm. In addition, the denser regions of the ADM-C (Figure 14) and AQD (Figure 16) have a much larger spread.

7 Conclusions

In this paper, we introduced a new, compact, purely quaternion based, closed-form solution to the absolute orientation problem. At the core of this research is the discovery of the linear relationship between the products of the components of the rotation quaternion and the input points themselves. The products can be expressed as a ratio of sums and difference of triple products.

The new closed-form was then extended, via a RANSAC-like framework, from a 3 point algorithm to a new, performance based n point algorithm. By quantifying the effects of error per individual estimate, the new algorithm is particularly becomes particularly resilient to mismatches. Simulations were used to analyze particular strengths and weaknesses of the new algorithm versus previous research. Many of the topics not fully covered here are discussed in greater detail in [12].

References

[1] R. Davies. newmat09. `http://nz.com/webnz/robert/nzc_nm09.html`. Software library.

[2] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, June 1981.

[3] J. D. Foley, A. vanDam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice*. Systems Programming Series. Addison Wesley, 2nd edition, 1990.

[4] W. R. Hamilton. *Elements of Quaternions*. Longmans, Green & Co., London, 1866.

[5] J. C. Hart, G. K. Francis, and L. H. Kauffman. Visualizing quaternion rotation. *ACM Transactions on Graphics*, 13(3):256–276, July 1994.

[6] B. K. P. Horn. *Robot Vision*. MIT Press, 1986.

[7] B. K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4):629–642, April 1987.

[8] B. K. P. Horn, H.M. Hilden, and S. Negahdaripour. Closed-form solution of absolute orientation using orthonormal matrices. *Journal of the Optical Society of America A*, 5(7):1127–1638, July 1988.

[9] T. Kanade, R. T. Collins, A. J. Lipton, P. Burt, and L. Wixson. Advances in cooperative multi-sensor video surveillance. In *Proceedings of the 1998 Image Understanding Workshop*, volume 1, pages 3–26, 1998.

[10] M. Kim, M. Kim, and S. Y. Shin. A general construction scheme for unit quaternion curves with simple high order derivatives. In *SIGGRAPH '95. Proceedings of the 22nd annual ACM conference on Computer graphics*, pages 369–376. Addison-Wesely, 1995.

[11] G. A. Korn and T.M. Korn. *Mathematical Handbook for Engineers and Scientists*. McGraw-Hill Book Co., New York, 1968.

[12] R. J. Micheals. A new closed-form approach to the absolute orientation problem. Master's thesis, Lehigh University, 1999.

[13] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipies in C: The Art of Scientific Computing*. Cambridge University Press, 2nd edition, 1992.

[14] R. Ramamoorthi. Fast construction of accurate quaterion splines. In *Proceedings of the 1997 ACM SIGGRAPH annual conference on Computer graphics*, pages 287–392, 1997.

[15] F. Sansò. An exact solution of the roto-translation problem. *Photogrammetria*, 29:203–216, 1973.

[16] M. Walker and L. Shao. Estimating 3-d location parameters using dual number quaternions. *CVGIP: Image Understanding*, 54(3):358–367, November 1991.

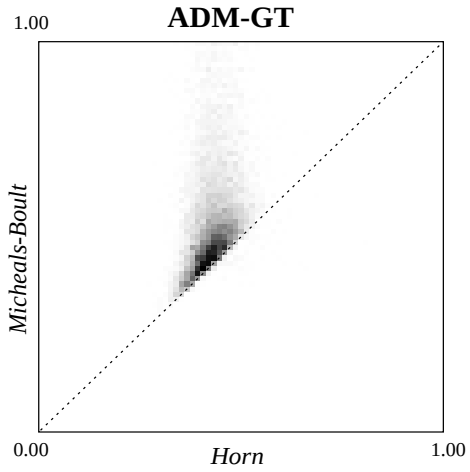

Figure 5. Normalized ADM-GT histogram. Point sets have moderate noise and are perfectly matched. *M-B* outperforms *Horn* in 2.81% of the trials.

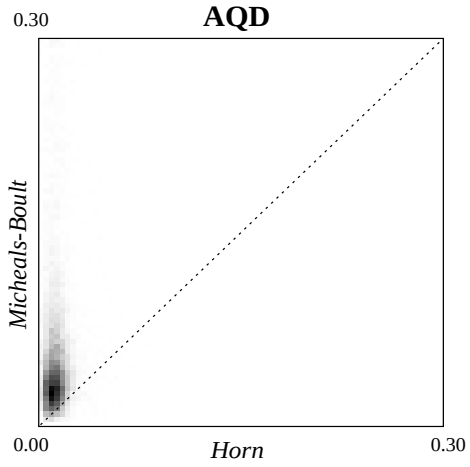

Figure 6. Normalized AQD histogram. Point sets have moderate noise and are perfectly matched. *M-B* outperforms *Horn* in 5.48% of the trials.

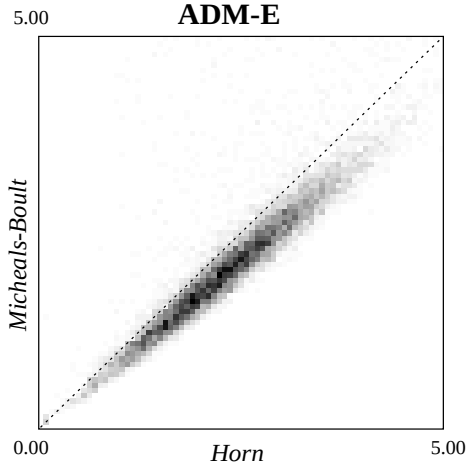

Figure 7. Normalized ADM-E histogram. Point sets have low noise & moderately high mismatch probability. *M-B* outperforms *Horn* in 90.83% of the trials.

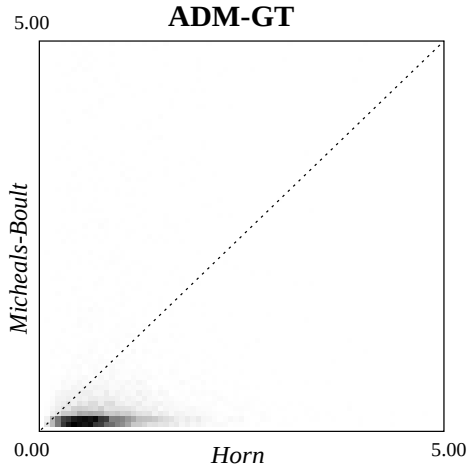

Figure 8. Normalized ADM-GT histogram. Point sets have low noise & moderately high mismatch probability. *M-B* outperforms *Horn* in 90.92% of the trials.

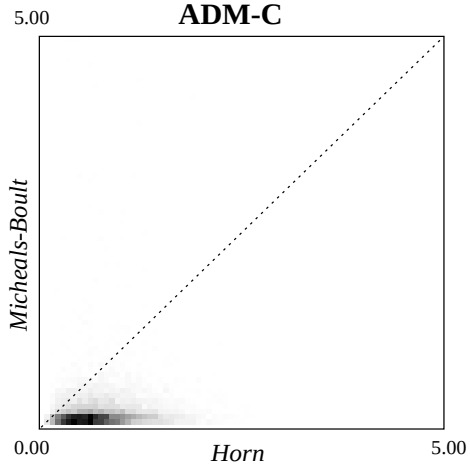

Figure 9. Normalized ADM-C histogram. Point sets have low noise & moderately high mismatch probability. *M-B* outperforms *Horn* in 90.81% of the trials.

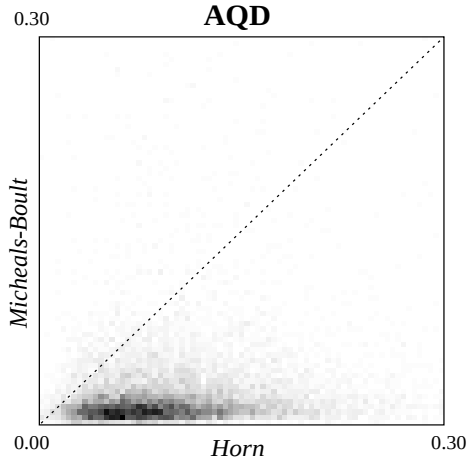

Figure 10. Normalized AQD histogram. Point sets have low noise & moderately high mismatch probability. *M-B* outperforms *Horn* in 92.32% of the trials.

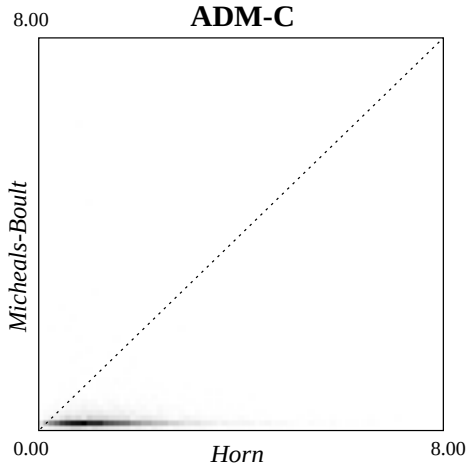

Figure 11. Normalized ADM-C histogram. Point sets have a known translation, low noise, and moderate outlier probability (10.0%). *M-B* outperforms *Horn* in 95.68% of the trials.

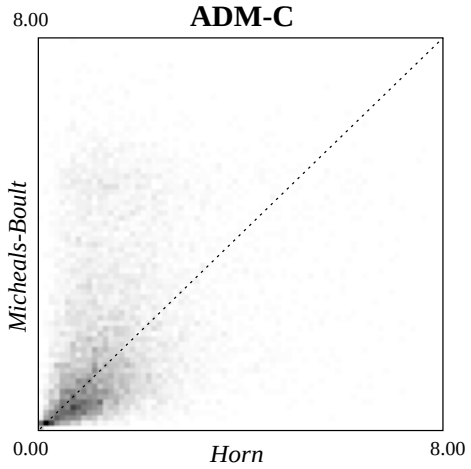

Figure 14. Normalized ADM-C histogram. Point sets have unknown translation, low noise, and moderate outlier probability (10.0%). *M-B* outperforms *Horn* in 45.44% of the trials.

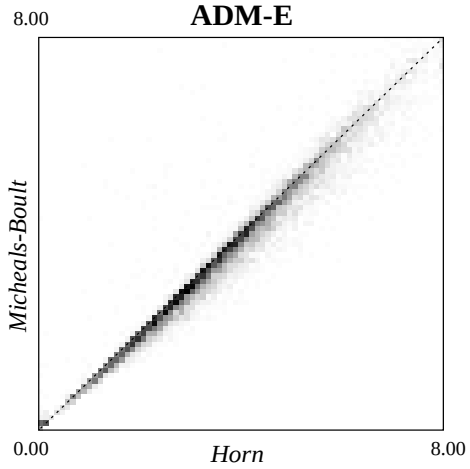

Figure 12. Normalized ADM-E histogram. Point sets have a known translation, low noise, and moderate outlier probability (10.0%). *M-B* outperforms *Horn* in 78.59% of the trials.

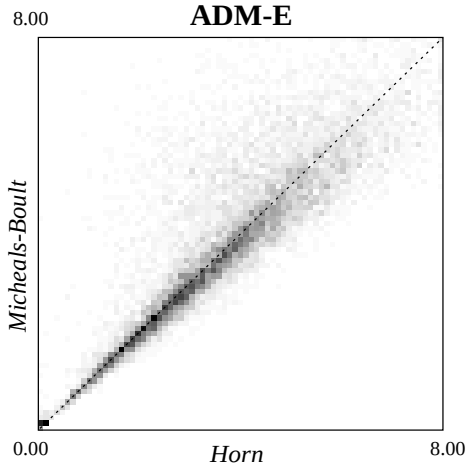

Figure 15. Normalized ADM-E histogram. Point sets have unknown translation, low noise, and moderate outlier probability (10.0%). *M-B* outperforms *Horn* in 53.26% of the trials.

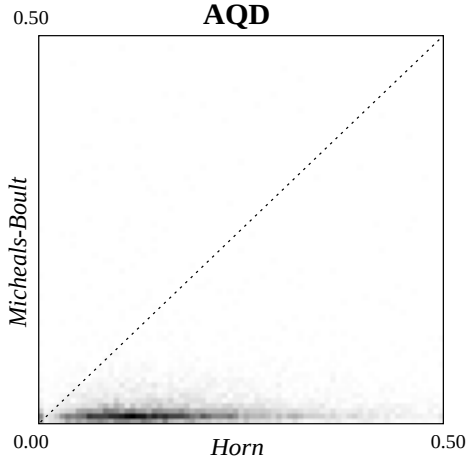

Figure 13. Normalized AQD histogram. Point sets have a known translation, low noise, and moderate outlier probability (10.0%). *M-B* outperforms *Horn* in 96.50% of the trials.

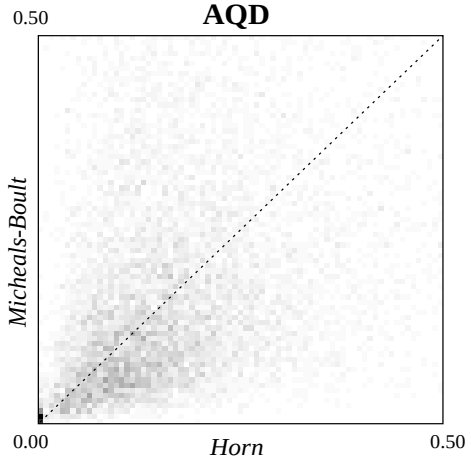

Figure 16. Normalized ADM-E histogram. Point sets have unknown translation, low noise, and moderate outlier probability (10.0%). *M-B* outperforms *Horn* in 50.89% of the trials.