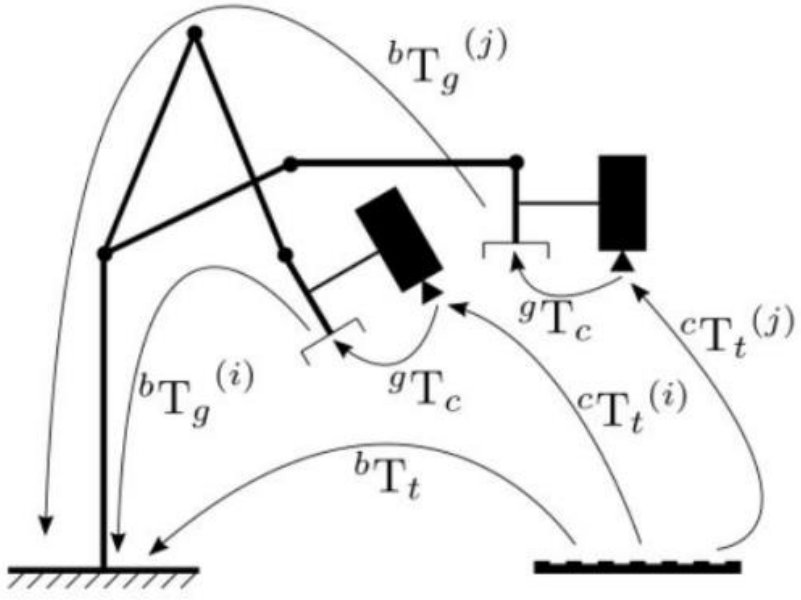
Hand Eye Calibration

http://campar.in.tum.de/Chair/HandEyeCalibration



We can solve the equations using loops:

$$^{b}T_{t} = {}^{b}T_{gj} * {}^{g}T_{c} * {}^{c}T_{tj}; \qquad {}^{b}T_{t} = {}^{b}T_{gi} * {}^{g}T_{c} * {}^{c}T_{ti}$$

$$\left({}^{b}T_{gj}^{-1} * {}^{b}T_{gi}\right) * {}^{g}T_{c} = {}^{g}T_{c} * \left({}^{c}T_{tj} * {}^{c}T_{ti}^{-1}\right)$$

Resulting in:

$$A * X = X * B, A = \begin{bmatrix} R_A & T_A \\ 0 & 1 \end{bmatrix}, B = \begin{bmatrix} R_B & T_B \\ 0 & 1 \end{bmatrix}, X = \begin{bmatrix} R_X & T_X \\ 0 & 1 \end{bmatrix}$$

There are two equations:

$$R_A * R_X = R_X * R_B$$

$$R_A * T_X + T_A = R_X * T_B + T_X$$

If R_X is solved, T_X is solved easily from the second equation. In order to solve this equation, we can do with linear algebra properties as R. Tsai, or as Y.C. Shiu, or by using quaternions.

Quaternions (https://en.wikipedia.org/wiki/Quaternion)

Extension of the complex numbers:

a) Complex numbers: $c = a + b * i;\ i^2 = -1$
b) Quaternions:

$$q = q_0 + q_1 * i + q_2 * j + q_3 * k; i^2 = j^2 = k^2 = -1; i * j = k; j * k = i; k * i = j$$

$q = (r, \vec{v}) = (q_0, [q_1, q_2, q_3])$ (represented as real part and vector part).

Quaternion multiplication table

	1	i	j	k
1	1	i	j	k
i	i	−1	k	−j
j	j	−k	−1	i
k	k	j	−i	−1

Quaternion addition is simply the some of its parts:

$$q + r = (q_0 + r_0) + (q_1 + r_1) * i + (q_2 + r_2) * j + (q_3 + r_3) * k$$

$$(r1, \overrightarrow{v1}) + (r2, \overrightarrow{v2}) = (r1 + r2, \overrightarrow{v1} + \overrightarrow{v2})$$

The product of quaternions is **non-commutative, but distributive**:

$$\boldsymbol{q * r =} (q_0 + q_1 * i + q_2 * j + q_3 * k) * (r_0 + r_1 * i + r_2 * j + r_3 * k) = (q_0, \vec{q}) * (r_0, \vec{r})$$

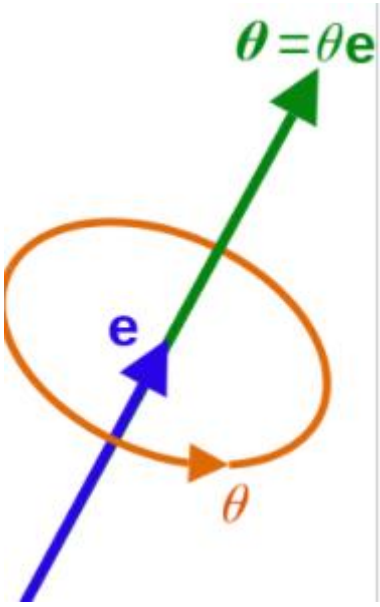
$$= (q_0 * r_0 - q_1 * r_1 - q_2 * r_2 - q_3 * r_3) + i * (q_0 * r_1 + q_1 * r_0 + q_2 * r_3 - q_3 * r_2) +$$

$$+ j * (q_0 * r_2 + q_2 * r_0 + q_3 * r_1 - q_1 * r_3) + k * (q_0 * r_3 + q_3 * r_0 + q_1 * r_2 - q_2 * r_1)$$

$$= (\boldsymbol{q_0 * r_0 - \vec{q}.\vec{r},\ q_0 * \vec{r} + r_0 * \vec{q} + \vec{q} \times \vec{r}})$$

Quaternions and 3d rotations

Rotations in 3D can be represented by 3x3 orthonormal matrices with determinant equal to one, by axis-angle or by unit quaternions. Axis angle representations correspond to a unitary vector *e*, and an angle θ, such that the original vectors (X_0, Y_0, Z_0) rotate around the axis *e*, by the angle θ, in order to find the new coordinate system (X_1, Y_1, Z_1). Unit Quaternions are quaternions q with $\boldsymbol{q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1}$.



The axis-angle representation is calculated as the product of the unitary *e* vector and the angle θ, resulting in a 3D vector representation of the rotation. As an example in Matlab, we define the eA vector as [1 2 3] / norm([1 2 3]), and the angle thA = wrapToPi(4) radians, in order to get the orthonormal matrix A:

```
eA = [1 2 3] / norm([1 2 3])
thA = wrapToPi(4)
A = axang2rotm([eA, thA])
eA =
    0.2673    0.5345    0.8018
thA =
   -2.2832
A =

   -0.5355    0.8430   -0.0502
   -0.3706   -0.1812    0.9110
    0.7589    0.5064    0.4094
```

```
rotm2axang(A)

ans =   -0.2673   -0.5345   -0.8018    2.2832
```

As we realize, the rotation representation can be an axis *e* and angle θ, or the opposite axis, *-e,* and opposite angle -θ, the product $e * \theta$ having the same value in both cases.

Unit quaternions are as well representations of 3d rotations, having the property that a unit quaternion and its opposite corresponds to the same matrix rotation. In addition, the quaternion representation of a rotation can be put in the following way:

$$q = \cos\left(\frac{\theta}{2}\right) + e * \sin\left(\frac{\theta}{2}\right)$$

Where e, θ corresponds to the axis-angle representation of the rotation.

The quaternions representing a vector or a point is defined as $(\mathbf{0}, \vec{\boldsymbol{v}})$ or $(\mathbf{0}, \boldsymbol{p})$**,** where the real part of the quaternion is 0, and the vector part of the quaternion corresponds to the coordinates of the vector or the point. The conjugate of the quaternion $q = (q_0, \vec{q}_v)$ is the quaternion with the same real part and the opposite vector part, i.e. $q^* = (q_0, -\vec{q}_v)$. The transformation of a vector by a matrix *R*, i.e. $\vec{v}_1 = R * \vec{v}$ is written with quaternions as:

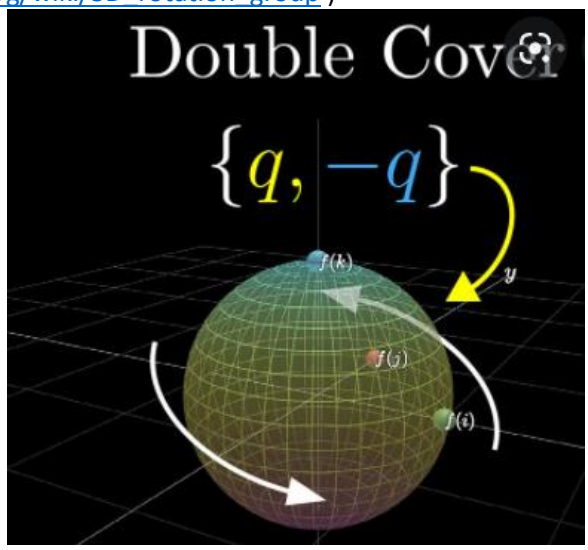
$$\boldsymbol{q}(\vec{\boldsymbol{v}}_1) = (\mathbf{0}, \vec{\boldsymbol{v}}_1) = \boldsymbol{q} * \boldsymbol{q}(\vec{\boldsymbol{v}}) * \boldsymbol{q}^* = \boldsymbol{q} * (\mathbf{0}, \vec{\boldsymbol{v}}) * \boldsymbol{q}^*$$

From this relation, it is easy to derive that the rotation matrix *R* can be written as:

$$R(q) = \begin{bmatrix} (q_0^2 + q_1^2 - q_2^2 - q_3^2) & 2 * (q_1 * q_2 - q_0 * q_3) & 2 * (q_1 * q_3 + q_0 * q_2) \\ 2 * (q_1 * q_2 + q_0 * q_3) & (q_0^2 + q_2^2 - q_1^2 - q_3^2) & 2 * (q_2 * q_3 - q_0 * q_1) \\ 2 * (q_1 * q_3 - q_0 * q_2) & 2 * (q_2 * q_3 + q_0 * q_1) & (q_0^2 + q_3^2 - q_1^2 - q_2^2) \end{bmatrix}$$

It is easy to verify that inverting the quaternion gives the same rotation matrix.

Topologically, the group of 3D rotations with determinant one (SO (3)), is homeomorphic to the unit hypersphere of the quaternion, where q and –q are connected.
https://en.wikipedia.org/wiki/Topology
(https://en.wikipedia.org/wiki/Algebraic_topology
https://en.wikipedia.org/wiki/Orthogonal_group,
https://en.wikipedia.org/wiki/3D_rotation_group)



The product of unit quaternions corresponds to the quaternion of the corresponding matrices.

```
X = axang2rotm([[3 1 0] / norm([3 1 0]), 2])
qA = rotm2quat(A);
qX = rotm2quat(X);
qAX = rotm2quat(A * X)
qAX1 = quatmultiply(qA, qX)
qAX =
   0.5482   0.3949  -0.7339  -0.0706
qAX1 =
   0.5482   0.3949  -0.7339  -0.0706
```

Lately, Matlab has created another class for quaternions:

help quaternion
quaternion - Create a quaternion array

A quaternion is a four-part hyper-complex number used in three-dimensional rotations and orientations.

quat = quaternion()
quat = quaternion(A,B,C,D)
quat = quaternion(matrix)
quat = quaternion(RV,'rotvec')
quat = quaternion(RV,'rotvecd')
quat = quaternion(RM,'rotmat',PF)
quat = quaternion(E,'euler',RS,PF)
quat = quaternion(E,'eulerd',RS,PF)

Documentation for quaternion

```
quatA = quaternion(A,'rotmat','point')
quatA =
  quaternion
    0.41615 - 0.24302i - 0.48604j - 0.72906k

quatX = quaternion(X,'rotmat','point')
quatX =
  quaternion
     0.5403 + 0.79829i +  0.2661j +       0k

quatA * quatX
ans =
  quaternion
     0.54818 +   0.3949i -  0.73387j - 0.070579k
```

In order to solve the solution of the previous equation, we can realize that:

$$R_A * R_X = R_X * R_{B;}\ R_A = R_X * R_B * R_x^{-1}$$

Then, R_A and R_B are similar (https://en.wikipedia.org/wiki/Matrix_similarity) and have similar eigenvalues, although the eigenvectors are different. The eigenvalues of an orthonormal matrix 1, $e^{i\theta} = \cos(\theta) + i * \sin(\theta), e^{-i\theta} = \cos(\theta) - i * \sin(\theta)$, where θ is the angle of rotation corresponding to the axis-angle representation. The eigenvector corresponding to the unit eigenvalue is the axis vector e.

```
B = X' * A * X;

% Eig of A and B should be similar as they are similar matrices
[VAA, eigA] = eig(A)
[VBB, eigB] = eig(B)
```

```
VAA =
  0.6814 + 0.0000i   0.6814 + 0.0000i   0.2673 + 0.0000i
 -0.1048 + 0.5883i  -0.1048 - 0.5883i   0.5345 + 0.0000i
 -0.1572 - 0.3922i  -0.1572 + 0.3922i   0.8018 + 0.0000i
eigA =
 -0.6536 + 0.7568i   0.0000 + 0.0000i   0.0000 + 0.0000i
  0.0000 + 0.0000i  -0.6536 - 0.7568i   0.0000 + 0.0000i
  0.0000 + 0.0000i   0.0000 + 0.0000i   1.0000 + 0.0000i
VBB =
  0.6888 + 0.0000i   0.6888 + 0.0000i  -0.2260 + 0.0000i
 -0.1080 - 0.5211i  -0.1080 + 0.5211i  -0.6584 + 0.0000i
  0.1177 - 0.4780i   0.1177 + 0.4780i   0.7179 + 0.0000i
eigB =
 -0.6536 + 0.7568i   0.0000 + 0.0000i   0.0000 + 0.0000i
  0.0000 + 0.0000i  -0.6536 - 0.7568i   0.0000 + 0.0000i
  0.0000 + 0.0000i   0.0000 + 0.0000i   1.0000 + 0.0000i
```

Besides that, the axes e_A, e_B corresponding to the axis-angle representation of matrices *A* and *B* are the eigenvectors of A and B, corresponding to the unit eigenvalues.

```
eA = rotm2axang(A)
eB = rotm2axang(B)
```

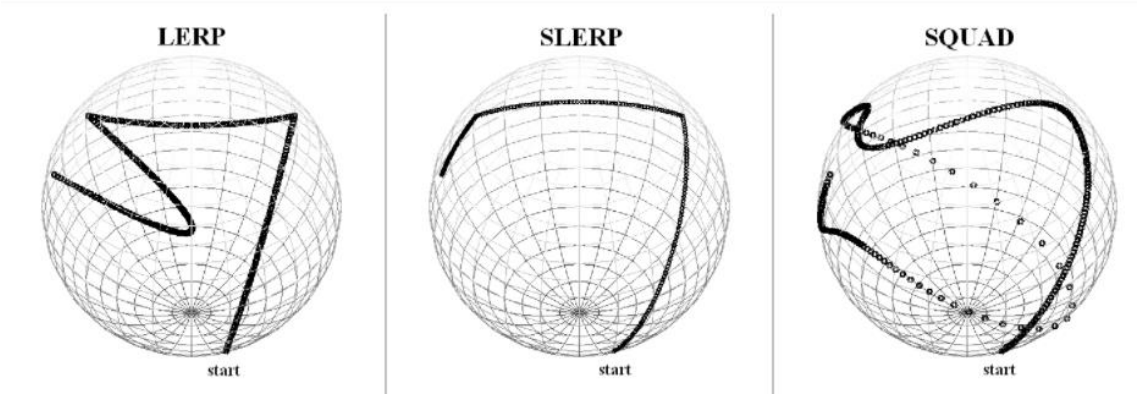
```
eA =  -0.2673  -0.5345  -0.8018   2.2832
eB =  -0.2260  -0.6584   0.7179   2.2832
```

Also, the axis $e_A = X * e_B$:

```
X * VBB(:, 3)
```

```
ans =
  -0.2673
  -0.5345
  -0.8018
```

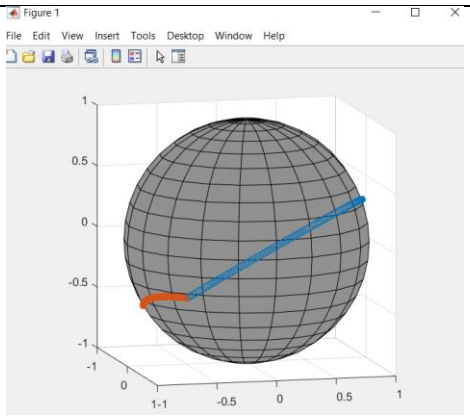
Finally, other advantages of quaternions are that the rotation interpolation is easier, using the so called Slerp (spherical linear interpolation, https://en.wikipedia.org/wiki/Slerp).



http://www.blackberry.com/developers/docs/6.0.0api/net/rim/device/api/math/Quaternion4f.html

Matlab Example

```
clear; clc; close all;

q1 = quaternion([75,-20,-10],'eulerd','ZYX','frame');
q2 = quaternion([-45,20,30],'eulerd','ZYX','frame');
q3 = quaternion([-90,30,60],'eulerd','ZYX','frame');

T = 0:0.01:1;
quats = slerp(q1,q2,T);
pts = rotatepoint(quats,[1 0 0]);

quats1 = slerp(q2,q3,T);
pts1 = rotatepoint(quats1,[1 0 0]);

figure
[X,Y,Z] = sphere;
surf(X,Y,Z,'FaceColor',[0.57 0.57 0.57])
hold on;

scatter3(pts(:,1),pts(:,2),pts(:,3))
scatter3(pts1(:,1),pts1(:,2),pts1(:,3))
view([69.23 36.60])
axis equal
```



Solution of the equation $\boldsymbol{R_A * R_X = R_X * R_B}$

The solution of this equation is simpler when we consider it as the products of their unit-quaternions:

$$q_A * q_X = q_X * q_B;\ q_A * q_X - q_X * q_B = 0$$

This last equation is a linear equation of the quaternion parameters of q_X, so that linear solution methods can be used. However, there are two possibilities for q_A and q_B, as the inverse of their quaternions is a possible solution, making the linear solution incongruent. In order to solve the signs of the quaternions q_A and q_B, we realize that $q_A = \cos\left(\frac{\theta_A}{2}\right) + e_A * \sin\left(\frac{\theta_A}{2}\right), q_B = \cos\left(\frac{\theta_B}{2}\right) + e_B * \sin\left(\frac{\theta_B}{2}\right)$, where $\theta_A = \theta_B = \theta$.

Note: Disregard the solutions where $\frac{\theta_A}{2}$ are close to $+\frac{\pi}{2}$ or $-\frac{\pi}{2}$, as $\cos\left(\frac{\theta_A}{2}\right)$ will be close to 0, and a small change in the angle produces a different sign. (very important in the simulation and real fitting).

Also, the product of two quaternions $\boldsymbol{q_1 * q_2} = (\boldsymbol{r_1}, \boldsymbol{\vec{v}_1}) * (\boldsymbol{r_2}, \boldsymbol{\vec{v}_2}) = (\boldsymbol{r_1 r_2 - \vec{v}_1.\vec{v}_2},\ \boldsymbol{r_1\vec{v}_2 + r_2\vec{v}_1 + \vec{v}_1 \times \vec{v}_2})$, where r_i and $\vec{v}_i$ are the real and vector part of q_i, $\vec{v}_1.\vec{v}_2$ is the dot product of the vector parts, and $\vec{v}_1 \times \vec{v}_2$ is the cross product of the vector parts. Then:

$$q_A * q_X = \left(\cos\left(\frac{\theta}{2}\right), e_A * \sin\left(\frac{\theta}{2}\right)\right) * \left(\cos\left(\frac{\theta_X}{2}\right), e_x * \sin\left(\frac{\theta_X}{2}\right)\right)$$
$$= \left(\boldsymbol{\cos\left(\frac{\theta}{2}\right) * \cos\left(\frac{\theta_X}{2}\right) - \sin\left(\frac{\theta}{2}\right) * \sin\left(\frac{\theta_X}{2}\right) * e_A.e_x}, vector\ part\right)$$

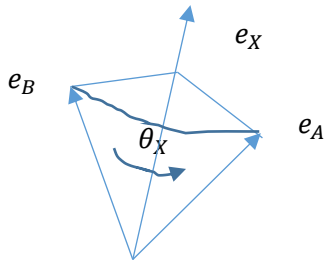
$$q_X * q_B = \left(\cos\left(\frac{\theta_X}{2}\right), e_x * \sin\left(\frac{\theta_X}{2}\right)\right) * \left(\cos\left(\frac{\theta}{2}\right), e_A * \sin\left(\frac{\theta}{2}\right)\right)$$
$$= \left(\boldsymbol{\cos\left(\frac{\theta}{2}\right) * \cos\left(\frac{\theta_X}{2}\right) - \sin\left(\frac{\theta}{2}\right) * \sin\left(\frac{\theta_X}{2}\right) * e_X.e_B}, vector\ part\right)$$

We realize also that $\boldsymbol{e_A.e_x = e_X.e_B}$ as $e_A = R_X * e_B$ (see Figure). The proof that $e_A = R_X * e_B$ is easy:

$R_A * e_A = e_A$, as e_A is the eigenvector corresponding to the unit eigenvector of R_A.

$$R_A = R_X * R_B * R_x^T$$

Thus, $R_A * e_A = e_A = R_X * R_B * R_x^T * e_A$, implying that $R_x^T * e_A = R_B * R_x^T * e_A$=> $e_B = R_X^T * e_A$



Thus the real parts of $\boldsymbol{q_A * q_X}$ and $\boldsymbol{q_X * q_B}$ should be the same. To do so, we have to choose q_A and q_B so that the real parts are the same.

```
% Compute the quaternions corresponding to A and B
% cos(theta/2) + v * sin(theta/2)

qA = rotm2quat(A)
qB = rotm2quat(B)
qX = rotm2quat(X)

% The product of rotations equal the product of quaternions
AX = A * X;
XB = X * B;

qAX = rotm2quat(AX)
qAX1 = quatmultiply(qA, qX)

qXB = rotm2quat(XB)
qXB1 = quatmultiply(qX, qB)

% The problem is if we use - qB
qXB2 = quatmultiply(qX, -qB)
```

```
qA =   0.4161  -0.2430  -0.4860  -0.7291
qB =   0.4161  -0.2055  -0.5987   0.6528
qX =   0.5403   0.7983   0.2661  -0.0000

qAX =   0.5482   0.3949  -0.7339  -0.0706
qAX1 = 0.5482   0.3949  -0.7339  -0.0706
qXB =   0.5482   0.3949  -0.7339  -0.0706
qXB1 =  0.5482   0.3949  -0.7339  -0.0706
qXB2 = -0.5482  -0.3949   0.7339   0.0706
```

Algorithm to calculate the hand eye X

The algorithm consists on taking a set of images of a calibration plate with a camera attached to the grip of the robot. The consecutive takes should have a difference in the rotation and translations. From each pair of images compute the equation $q_A * q_X - q_X * q_B = 0,\ C * q_X = 0,$ with the constraint that the real parts of q_A and q_B have the same sign (if not change the sign of q_A). Solve linearly q_X, so that the rotation part R_X is solved, and with it solve the translational part.

$$q_A * q_X = (q_{A0} * q_{X0} - q_{A1} * q_{X1} - q_{A2} * q_{X2} - q_{A3} * q_{X3}) +$$
$$i * (q_{A0} * q_{X1} + q_{A1} * q_{X0} + q_{A2} * q_{X3} - q_{A3} * q_{X2}) +$$
$$j * (q_{A0} * q_{X2} + q_{A2} * q_{X0} + q_{A3} * q_{X1} - q_{A1} * q_{X3}) +$$
$$k * (q_{A0} * q_{X3} + q_{A3} * q_{X0} + q_{A1} * q_{X2} - q_{A2} * q_{X1})$$

$$q_X * q_B = (q_{B0} * q_{X0} - q_{B1} * q_{X1} - q_{B2} * q_{X2} - q_{B3} * q_{X3}) +$$
$$i * (q_{X0} * q_{B1} + q_{X1} * q_{B0} + q_{X2} * q_{B3} - q_{X3} * q_{B2}) +$$
$$j * (q_{X0} * q_{B2} + q_{X2} * q_{B0} + q_{X3} * q_{B1} - q_{X1} * q_{B3}) +$$
$$k * (q_{X0} * q_{B3} + q_{X3} * q_{B0} + q_{X1} * q_{B2} - q_{X2} * q_{B1})$$

$$\begin{bmatrix} (q_{A0} - q_{B0}) & -(q_{A1} - q_{B1}) & -(q_{A2} - q_{B2}) & -(q_{A3} - q_{B3}) \\ (q_{A1} - q_{B1}) & (q_{A0} - q_{B0}) & -(q_{A3} + q_{B3}) & (q_{A2} + q_{B2}) \\ (q_{A2} - q_{B2}) & (q_{A3} + q_{B3}) & (q_{A0} - q_{B0}) & -(q_{A1} + q_{B1}) \\ (q_{A3} - q_{B3}) & -(q_{A2} + q_{B2}) & (q_{A1} + q_{B1}) & (q_{A0} - q_{B0}) \end{bmatrix} * \begin{bmatrix} q_{X0} \\ q_{X1} \\ q_{X2} \\ q_{X3} \end{bmatrix} = \vec{0}$$

$$(q_{A0} - q_{B0}) * I_{4x4} + \begin{bmatrix} 0 & -(\vec{q}_A - \vec{q}_B)^T \\ (\vec{q}_A - \vec{q}_B) & skew(\vec{q}_A + \vec{q}_B) \end{bmatrix} = \vec{0}$$

First, make a simulation of the system, with a simulated camera and several poses of the camera pointing to the center of a calibration plate. Then, make a real experiment.

Calculation of the transformation between the plate and the base of the robot $^{b}T_{t}$

$$^{b}T_{t} = {}^{b}T_{gj} * {}^{g}T_{c} * {}^{c}T_{tj}; \qquad {}^{b}T_{t} = {}^{b}T_{gi} * {}^{g}T_{c} * {}^{c}T_{ti}$$

$$^{g}T_{c} = {}^{b}T_{gj}^{-1} * {}^{b}T_{t} * {}^{c}T_{tj}^{-1}; \qquad {}^{g}T_{c} = {}^{b}T_{gi}^{-1} * {}^{b}T_{t} * {}^{c}T_{ti}^{-1}$$

Resulting in:

$$^{b}T_{gj}^{-1} * {}^{b}T_{t} * {}^{c}T_{tj}^{-1} = {}^{b}T_{gi}^{-1} * {}^{b}T_{t} * {}^{c}T_{ti}^{-1}$$

$$({}^{b}T_{gi} * {}^{b}T_{gj}^{-1}) * {}^{b}T_{t} = {}^{b}T_{t} * ({}^{c}T_{ti}^{-1} * {}^{c}T_{tj})$$

Being another system $A1 * Y = Y * B1$, that resolves in the same previous manner.

Appendix1 Symbolic matrix R calculation as function of quaternion

It is based on the formula $\boldsymbol{q}(\vec{\boldsymbol{v}}_1) = (\boldsymbol{0}, \vec{\boldsymbol{v}}_1) = \boldsymbol{q} * \boldsymbol{q}(\vec{\boldsymbol{v}}) * \boldsymbol{q}^* = \boldsymbol{q} * (\boldsymbol{0}, \vec{\boldsymbol{v}}) * \boldsymbol{q}^*$

```
q = sym('q', [1, 4], 'real');
qconj = [q(1), -q(2:end)];
v = sym('v', [1, 4], 'real');
v(1) = 0;

qvqconj = simplify(myquatmultiply(myquatmultiply(q, v), qconj));

for i = 1:3
  for j = 1:3
    R(i, j) = simplify(diff(collect(qvqconj(i + 1), v(j + 1)), v(j +
1)));
  end
end

disp(R)
%%
function s = myquatmultiply(q, r)
  q0 = q(1);
  r0 = r(1);
  qv = q(2:end);
  rv = r(2:end);

  s0 = q0 * r0 - qv * rv';
  sv = q0 * rv + r0 * qv + cross(qv, rv);

  s = [s0, sv];
end
```

ExampleQuaternionToMatrix

```
[q1^2 + q2^2 - q3^2 - q4^2,       2*q2*q3 - 2*q1*q4,           2*q1*q3 + 2*q2*q4]
[     2*q1*q4 + 2*q2*q3,    q1^2 - q2^2 + q3^2 - q4^2,        2*q3*q4 - 2*q1*q2]
[     2*q2*q4 - 2*q1*q3,        2*q1*q2 + 2*q3*q4,     q1^2 - q2^2 - q3^2 + q4^2]
```

For the proof of $q = \cos\left(\frac{\theta}{2}\right) + e * \sin\left(\frac{\theta}{2}\right)$, see Chapter 15 of Galliers book. It is rather complicated.