Recreation of Robot Sensor Calibration: Solving AX=XB on the Euclidean Group

Priya Gillan
Mechanical Engineering
The Johns Hopkins University
Baltimore, USA
pgillan2@jh.edu

***Abstract*—Hand-eye calibration is used to find the transformation between a coordinate frame on the sensor of a robot and a coordinate frame on the robot itself. Usually, the sensor on the robot is on the end effector link. The equation AX=XB is used to calibrate the robot hand with the sensor.**

I. Introduction

AX=XB is on the Euclidean group. We can use methods from Lie algebra and Lie theory to derive AX=XB. Usually, A and B are known, and X is unknown. A and B are n by n matrices and are found by calculating the forward kinematics of the UR5 from the base to hand, which is described by E_1 and E_2 as illustrated in [Fig.1], and the measurement from the camera to the calibration board, which is described by S_1 and S_2 also illustrated in [Fig.1]. X is used to determine the precise location of the sensor.

Matrix A describes the position and orientation of the frame located on the wrist of the UR5 in relation to itself after a movement. Matrix B is describing the position and orientation of the sensor frame in relation to itself after a movement. Matrix X describes the position and orientation of the sensor frame in relation to the wrist frame.

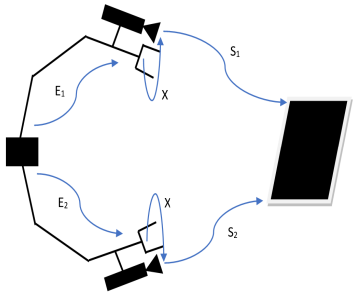

Fig. 1. The UR5 robot is moved to two positions E_1 and E_2. From those positions, we can also collect S_1 and S_2 which are the positions of the sensors in relation to the calibration board. X is the measurement from the end effector to the sensor.

II. Lie Algebra

SE(3) is the Euclidean group of rigid-body motions. Those matrices can be characterized as follows:

$$\begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \in \text{SE(3)}$$

Where R is a rotation matrix in SO(3) and t is a 3x1 translation vector. Since the Euclidean group is also a Lie group, Lie algebra can be used to derive AX=XB.

The Lie algebra associated with SO(3) is denoted by so(3). For example:

$$\begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \in \text{SO(3)}$$

By taking the "V" operation of the matrix above we get:

$$\begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}^V = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} = [\omega] \in \text{so(3)}$$

Given the definition above, we can show se(3) as:

$$\begin{bmatrix} [\omega] & v \\ 0 & 1 \end{bmatrix} \in \text{se(3)}$$

One fundamental part of Lie groups is exponential mapping. Given the Lie group **G**, and the corresponding Lie algebra **g**, the exponential mapping is:

$$\mathbf{g} \rightarrow \mathbf{G}$$

Matrix exponential is given by:

$$\exp(A) = I + A + \tfrac{1}{2!}A^2 + \dots$$

where A $\in$ **g**.

Exponential mapping for so(3) to SO(3) is given by lemma 1.
Lemma 1: Given $[\omega] \in$ so(3), exp$[\omega]$ is an element of SO(3)

$$\exp[\omega] = I + \tfrac{\sin\|\omega\|}{\|\omega\|}[\omega] + \tfrac{1-\cos\|\omega\|}{\|\omega\|^2}[\omega]^2$$

Proof of lemma 1: Since $\exp[\omega] \in \text{SO(3)}$, we can use the characteristic polynomial, which is

$$s^3 + \|\omega\|^2 s$$

Then using the Cayley-Hamilton Theorem,

$$[\omega]^3 = -\|\omega\|^2[\omega]$$

By lemma 1, SO(3) can be visualized as a 3D solid ball with a radius of π and centered at the origin. The point ω, in the ball, represents a rotation by the angle $\|\omega\|$ radians. This exponential map gives us the local coordinates for the set of rotation matrices.

Next, we will discuss the inverse of the exponential map also known as logarithm. **Lemma 2:** Let R ∈ SO(3) such that Tr(R) $\neq$ 1.

$$\log R = \frac{\phi}{2\sin\phi}\left(R - R^T\right)$$

$$\text{where } \phi \text{ satisfies } 1 + 2\cos\phi = \mathrm{Tr}(R),$$

$$|\phi| < \pi, \text{ and } \|\log R\|^2 = \phi^2$$

Proof of lemma 2: Using Euler's Theorem, for any $R \in SO(3)$, there exists a $Q \in SO(3)$ and $0 \leq \phi \leq 2\pi$.

$$R = Q \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} Q^T$$

The eigenvalues of R are 1, and $e^{\pm i\phi}$. $\mathrm{Tr}(R) \neq 1$ also shows that $\phi \neq \pi$.

The log formula gives us a point in the solid ball of radius π that corresponds to a particular rotation. log(R) has two solutions.

$$\log(R) = \pm\pi[\hat{\omega}]$$

We can now derive the equation for the exponential of SE(3). **Lemma 3:** Let [ω] ∈ SO(3) and v ϵ $\mathbb{R}^3$

$$\exp\begin{bmatrix} [\omega] & v \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} \exp[\omega] & Av \\ 0 & 0 \end{bmatrix}$$

where A can be expressed like in Lemma 1.

$$A = I + \frac{1-\cos(\|\omega\|)}{\|\omega\|^2}[\omega] + \frac{\|\omega\| - \sin(\|\omega\|)}{\|\omega\|^3}[\omega]^2$$

Proof of lemma 3: Writing out the series expansion:

$$exp\begin{bmatrix} [\omega] & v \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} \exp[\omega] & \left(\sum_{n=1}^{\infty} \frac{[\omega]^{n-1}}{n!}\right) v \\ 0 & 1 \end{bmatrix}$$

This also gives a set of local coordinates for all of SE(3).

Next, is deriving the equations for the logarithm of SE(3). **Lemma 4:** Let R ∈ SO(3) such that Tr(R) $\neq$ -1, and t ∈ $\mathbb{R}^3$.

$$\log\begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} [\omega] & A^{-1}t \\ 0 & 0 \end{bmatrix}$$

where [ω] = log(R) and

$$A^{-1} = I - \frac{1}{2} \times [\omega] + \frac{2sin||\omega|| - ||\omega||(1+cos||\omega||)}{2||\omega||^2 sin||\omega||} \times ||\omega||^2$$

Proof of lemma 4: Using the Cayley-Hamilton Theorem,

$$[\omega]^3 = -||\omega||^2 [\omega]$$

A^{-1} is a quadratic matrix polynomial in [ω].

III. Solution to AX=XB

We find an A and a B by carrying out the following calculations:

$$A = E_1^{-1} E_2$$

$$B = S_1 S_2^{-1}$$

To write AX=XB in matrix form we can denote A, B, and X in the following form.

$$A = \begin{bmatrix} R_A & t_A \\ 0 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} R_B & t_B \\ 0 & 1 \end{bmatrix}$$

$$X = \begin{bmatrix} R_X & t_X \\ 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} R_A & t_A \\ 0 & 1 \end{bmatrix}\begin{bmatrix} R_X & t \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R_X & t_X \\ 0 & 1 \end{bmatrix}\begin{bmatrix} R_B & t_B \\ 0 & 1 \end{bmatrix}$$

The problem can be broken down into the rotation part and the translation part. When we are using the closed-form solution the two parts can be described:

$$R_A R_X = R_X R_B \ (Rotation)$$

$$R_A t_X + t_A = R_X t_B + t_X \ (Translation)$$

To solve the rotation, we define two alphas and two betas. Those can be found by doing the following computation:

$$\alpha_1 = \log(R_{A1})^V$$

$$\alpha_2 = \log(R_{A2})^V$$

$$\beta_1 = \log(R_{B1})^V$$

$$\beta_2 = \log(R_{B2})^V$$

We can then build two matrices:

$$\mathcal{A} = [\alpha_1 \quad \alpha_2 \quad \alpha_1 \times \alpha_2]$$

$$\mathcal{B} = [\beta_1 \quad \beta_2 \quad \beta_1 \times \beta_2]$$

To solve the rotation in X we can use the new $\mathcal{A}$ and $\mathcal{B}$ matrices.

$$R_X = \mathcal{A}\mathcal{B}^{-1} \tag{1}$$

Once we have the rotation for the matrix X, we can solve for the translation using the least squares method.

$$\begin{bmatrix} R_{A_1} - I \\ R_{A_2} - I \end{bmatrix} t_X = \begin{bmatrix} R_X t_{B_1} - t_{A_1} \\ R_X t_{B_2} - t_{A_2} \end{bmatrix} \tag{2}$$

When we have more than two data points, the problem gets a little more complicated. We can collect the alphas and betas the same way we did when we only had two data points, however, we use polar decomposition to solve R_X.

$$R_X = (M^T M)^{\frac{-1}{2}} M^T$$

$$M = \sum_{i=1}^{N} \beta_i \alpha_i^T$$

$$(M^T M)^{\frac{-1}{2}} = Q \begin{bmatrix} \frac{1}{\sqrt{\lambda_1}} & 0 & 0 \\ 0 & \frac{1}{\sqrt{\lambda_2}} & 0 \\ 0 & 0 & \frac{1}{\sqrt{\lambda_3}} \end{bmatrix} Q^{-1}$$

Once we have R_X, we can solve for t_X by using least squares method.

IV. Code

Using MATLAB, ROS, and RVIZ I was able to simulate the process of solving for the X matrix of AX=XB. This was done as an assignment for Algorithms for Sensor-Based Robots at

Johns Hopkins University, but I adjusted some things for the purpose of this paper.

Starting with the closed-form solution, the MATLAB script takes in the forward kinematics (E_{BH}) of the UR5 from the base to the hand of the robot. It also takes in an E_{SC} which is the measurement from the sensor to the calibration board. Both E_{BH} and E_{SC} are 3x7 matrices. Each matrix contains 3 1x7 vectors in the form of [t_x t_y t_z q_x q_y q_z q_ω] where t is the translation and q is a quaternion. One thing to note is ROS and MATLAB format quaternions differently. MATLAB formats quaternions: [q_ω q_x q_y q_z] and ROS formats quaternions: [q_x q_y q_z q_ω]. To handle this, I did the following:

```
r_ebh1=quat2rotm([e_bh(1,7) e_bh(1,4)
    e_bh(1,5) e_bh(1,6)]);
t_ebh1=[e_bh(1,1); e_bh(1,2); e_bh(1,3)];
ebh1=[rebh1 tebh1; 0 0 0 1];

r_esc1=quat2rotm([e_sc(1,7) e_sc(1,4)
    e_sc(1,5) e_sc(1,6)]);
t_esc1=[e_sc(1,1); e_sc(1,2); e_sc(1,3)];
esc1=[resc1 tesc1; 0 0 0 1];
```

This reformat the quaternions in a way that MATLAB could understand and change them into matrices. The lines of code above created two 4x4 matrices.

I repeated this process two additional times to get three E_{BH} matrices and three E_{SC} matrices. Then to calculate A and B I used the following lines of code:

```
A1=ebh1\ebh2;
A2=ebh2\ebh3;

B1=esc1/esc2;
B2=esc2/esc3;
```

This gives me two A matrices and two B matrices to calculate necessary α's and β's

From there I need to extract the rotation and translation parts from both A's and both B's to find the alphas and betas needed to calculate the rotation portion of X.

```
Ra=A1(1:3,1:3);
ta=A1(1:3, 4);

Rb=B1(1:3,1:3);
tb=B1(1:3, 4);

alpha1=InvSkew(logm(Ra));
beta1=InvSkew(logm(Rb1));
```

The InvSkew() method takes the "V" operation of a matrix. I repeat this process with A2 and B2 to get alpha2 and beta2.

```
A=[alpha1 alpha2 cross(alpha1, alpha2)];
B=[beta1 beta2 cross(beta1,beta2)];

%% calculate Rx
Rx=(A/B);

%% set up matrices to calculate
%% least squares
ls1=[Ra1-eye(3); Ra2-eye(3)];
ls2=[Rx*tb1-ta1; Rx*tb2-ta2];

%% calculate least sqaures to find
%% translation
tx=lsqr(ls1,ls2);

%% assemble X
X=[Rx tx; 0 0 0 1];
```

The code above is the implementation of of [eq. (1)] and [eq. (2)]. Now we have an X and have successfully calibrated the sensor for the closed-form solution.

When we have more than 2 A's and B's, we have to change how we solve for X. If we have n E_{BH}s and n E_{SC}s then we will have n-2 A's and B's. To get all the A's and all the B's, I do the following:

```
for i=1:n-2
  reb1=quat2rotm([e_bh(i,7) e_bh(i,4)
    e_bh(i,5) e_bh(i,6)]);
  teb1=[e_bh(i,1); e_bh(i,2);
    e_bh(i,3)];
  reb2=quat2rotm([e_bh(i+1,7) e_bh(i+1,4)
    e_bh(i+1,5) e_bh(i+1,6)]);
  teb2=[e_bh(i+1,1); e_bh(i+1,2);
    e_bh(i+1,3)];
  reb3=quat2rotm([e_bh(i+2,7) e_bh(i+2,4)
    e_bh(i+2,5) e_bh(i+2,6)]);
  teb3=[e_bh(i+2,1); e_bh(i+2,2);
    e_bh(i+2,3)];

  res1=quat2rotm([e_sc(i,7) e_sc(i,4)
    e_sc(i,5) e_sc(i,6)]);
  tes1=[e_sc(i,1); e_sc(i,2);
    e_sc(i,3)];
  res2=quat2rotm([e_sc(i+1,7) e_sc(i+1,4)
    e_sc(i+1,5) e_sc(i+1,6)]);
  tes2=[e_sc(i+1,1); e_sc(i+1,2);
    e_sc(i+1,3)];
  res3=quat2rotm([e_sc(i+2,7) e_sc(i+2,4)
    e_sc(i+2,5) e_sc(i+2,6)]);
  tes3=[e_sc(i+2,1); e_sc(i+2,2);
    e_sc(i+2,3)];

    ebh1=[reb1 teb1; 0 0 0 1];
    ebh2=[reb2 teb2; 0 0 0 1];
    ebh3=[reb3 teb3; 0 0 0 1];

    esc1=[res1 tes1; 0 0 0 1];
    esc2=[res2 tes2; 0 0 0 1];
    esc3=[res3 tes3; 0 0 0 1];

    a1=ebh1\ebh2;
```

```
    ra1=a1(1:3,1:3);
    ta1=a1(1:3,4);

    a2=ebh2\ebh3;
    ra2=a2(1:3,1:3);
    ta2=a2(1:3,4);

    b1=esc1/esc2;
    rb1=b1(1:3,1:3);
    tb1=b1(1:3, 4);

    b2=esc2/esc3;
    rb2=b2(1:3,1:3);
    tb2=b2(1:3,4);

    alpha1=InvSkew(logm(ra1));
    alpha2=InvSkew(logm(ra2));
    beta1=InvSkew(logm(rb1));
    beta2=InvSkew(logm(rb2));

    alphas=[alphas alpha1 alpha2];
    betas=[betas beta1 beta2];

    RA=[RA ra1 ra2];
    tA=[tA ta1 ta2];
    RB=[RB rb1 rb2];
    tB=[tB tb1 tb2];
end
```

The above code calculates all the A's and B's and their corresponding α's and β's.

I then can call the solveRx() and the solveTx() functions to assemble the X matrix.

```
RX=solveRx(alphas, betas);
tX=solveTx(RA, tA, RB, tB, RX);
X=[RX tX; 0 0 0 1];
```

The solveRx() function looks like this and gives us the rotation matrix of X:

```
M=zeros(3,3);
for i=1:b
    M=M+(betas(1:3,i)*
        transpose(alphas(1:3,i)));
end
X=transpose(M)*M;
e=eig(X);
[Q,V]=eig(X);
lambda=[1/sqrt(e(1)) 0 0;
        0 1/sqrt(e(2)) 0;
        0 0 1/sqrt(e(3))];

Rx=(Q*lambda/Q)*transpose(M);
```

And the solveTx() looks like this and hives us the translation vector of X:

```
[a,b,n]=size(tA);
x=[];
y=[];

for i=1:b
    x(i*3-2:i*3,1:3)=eye(3)-
        RA(1:3,i*3-2:i*3);
    y(i*3-2:i*3,1)=tA(1:3,i)-RX*tB(1:3,i);
end
tx=lsqr(x,y);
```

V. Simulation Data

In RVIZ I was able to collect simulation data points to run the hand-eye calibration code. Using RVIZ, I positioned the

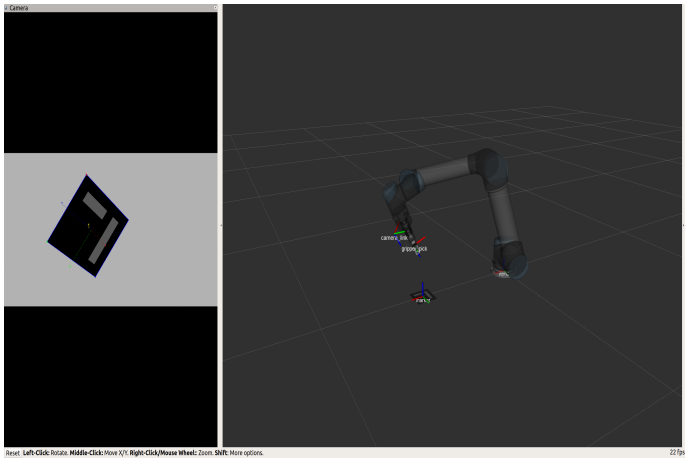

Fig. 2. The left panel shows the camera view and you can see the AR tag in with a coordinate frame. The right panel shows the configuration of the robot.

UR5 differently 15 times and collect 15 E_{BH}'s and 15 E_{SC}'s. In the Linux terminal I used this command to get the E_{BH}:

```
rosrun tf tf_echo world gripper_pick
```

Then to get the E_{SC} I run:

```
rosrun tf tf_echo camera_link marker
```

To get the actual X matrix I run:

```
rosrun tf tf_echo gripper_pick camera_link
```

In my case that yields this matrix:

$$X = \begin{bmatrix} 0.6470 & 0.7567 & 0.0936 & 0.0590 \\ -0.7605 & 0.6493 & -0.0074 & 0.0210 \\ -0.0552 & 0.0760 & 0.9956 & 0.1170 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

After running the closed-form solution with 3 points I got this matrix:

$$X_{cf} = \begin{bmatrix} 0.6577 & 0.3124 & 0.0620 & -0.0701 \\ -0.5578 & 0.6819 & -0.0263 & 0.0931 \\ -0.0005 & -0.1003 & 0.6617 & -0.1642 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

After running the open-form solution with 15 points I got this matrix:

$$X_{of} = \begin{bmatrix} 0.6837 & 0.7276 & 0.0565 & -0.0634 \\ -0.7236 & 0.6859 & -0.0766 & 0.0370 \\ -0.0945 & -0.0115 & 0.9955 & -0.1152 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

After comparing the X with X_{cf}, the error is 0.0298. The error when comparing X with X_{of} is 0.0046.

VI. Conclusion

Lie algebra is a convenient way to derive AX=XB. The closed-form solution is less accurate compared to collecting multiple configurations. This was shown in my simulation. Obviously, the more points collected, the more accurate the measurement will be. Another important thing to note is the accuracy of the measurement is affected by noise. capturing configurations far away from the calibration board creates more noise and will yield a less accurate X matrix. It is best to collect configurations as close to the configuration board as possible.

References

[1] F. C. Park and B. J. Martin, "Robot sensor calibration: SOLVING AX=XB on the Euclidean group," IEEE Transactions on Robotics and Automation, vol. 10, no. 5, pp. 717–721, Oct. 1994.

[2] G. D. Hager and S. Leonard, "Hand-eye calibration - cs.jhu.edu." [Online]. Available: https://www.cs.jhu.edu/~sleonard/lecture03.pdf. [Accessed: 06-May-2023].

[3] "Euclidean group," Wikipedia, 21-Sep-2022. [Online]. Available: https://en.wikipedia.org/wiki/Euclidean_group. [Accessed: 06-May-2023].