Appendix A: Documentation for Calibration Pipeline

This appendix explains how the main sub functions such as hand-eye calibration or polynomial fitting were approached and implemented. The calibration pipeline can be run by running the script *galen_calibration_virtual.m*, which calls in the necessary subfunctions to first perform hand-eye calibration on the robot and the tool attached to it, and then fits and applies a correction polynomial. This document provides the details of each subfunction and explains how they come together in *galen_calibration_virtual.m.* Refer to the wiki or Final Project Report for referenced equations.

Pipeline Summary

- **File Name:** galen_calibration.pipeline.m
- **Input:** none, but data reading files to fit the polynomial and to test the correction named get_data_1.m and get_test_data_2.m respectively are used. They input the 3D arrays of homogeneous transformation matrices C and K, as well as ρ (k_{dof}).
- **Output:** S, the correction polynomial coefficients, old_error_test, error observed in test data before calibration, and norm_error_test, error observed after calibration.

1. For both trajectories, acquire C, K, k_{dof} from the csv files using *get_data_1.m*.
2. Calculate the A and B matrices for the hand-eye calibration problem as described in Equations (5) and (8) using the data from the wrist trajectory. This is because having different rotations while collecting the experimental data results in a better, more robust solution for the AX=XB problem.
3. Solve the AX=XB problem for T_{tip} and T_B using *axxb.m.*
4. For each pose in both trajectories, calculate K_{pre} as in Equation (9) and ΔK_i as in Equation (10).
5. Convert ΔK_i to DOF as e_{dof}, using the *skew2vector.m* function.
6. Fit and apply a Bernstein polynomial to map k_{dof} to e_{dof}
7. Acquire C_{test}, K_{test}, $k_{dof,\ test}$ for the test dataset csv files using *get_test_data_2.m.*
8. Apply the correction polynomial to $k_{dof,\ test}$ to calculate $e_{bp,dof}$
9. Convert $e_{bp,dof}$ to $\Delta K_{BP,\ test,\ i}$ and calculate $K_{new,\ test}$ as Equation (14)
10. Calculate error between $K_{new,\ test}$, K_{test} , and $K_{pre,test}$ to quantify calibration performance.

Main Subfunctions

Solver for the AX = XB Problem

- **File Name:** axxb.m
- **Input:** A and B, two 4x4xN matrices that contain N 4x4 homogeneous transformation matrices. For solving the base transformation T_B, $(C_i C_0^{-1})T_B = T_B(K_i K_0^{-1})$, A = $(C_i C_0^{-1})$ and B = $(K_i K_0^{-1})$. For solving the tool tip transformation T_{tip}, $(C_0^{-1} C_i)T_{tip}^{-1} = T_{tip}^{-1}(K_0^{-1} K_i)$, A = $(C_0^{-1} C_i)$ and B = $(K_0^{-1} K_i)$.
- **Output:** The unknown transformation matrix X

"axxb.m" uses the Park and Martin hand-eye-calibration method to solve for an unknown transformation inherent to two frame transformation paths. The procedure is summarized for a robotic system in the following steps:

1. The first transformation path is the robot's forward kinematics up until the base frame of the unknown transformation. The second transformation path is the transformation from the end of the unknown transformation to some reference object whose transformation is known.
2. Move the robot to several different poses (>=3) and record the transformations for both paths.
3. For each of the two paths, find the relative transformation between each data point and a singular transformation in the data set. For the robot forward kinematics path, this given the relative transformation between the base of the unknown transformation between two different robot poses. For the reference object path, this gives the relative transformation between the ends of the unknown transformation between two paths.
4. Use a polar decomposition to solve for the unknown transformation's rotational component.
5. Use a least-squares solution to solve for the unknown transformation's translational component.

A more detailed explanation of this process can be found in References [4] and [9] of the Final Project Report.

Ground Truth Forward Kinematics

- **File Name:** forwardKinematics.m
- **Input:** A 5x1 vector, elements of which are the joint configurations for which we want the forward kinematics
- **Output:** Cartesian position of the end efector, and the tilt and pitch angles
- **Notes:** Provided by Galen Robotics

Function for Kinematic Correction

- **File Name:** correct.m
- **Input:** $\mathbf{C}^{data}_{2D}$, and $\mathbf{C}^{cor}_{2D}$, 2D compilations of each C vector measured and corrected respectively.
- **Output:** The matrix **S** that stores the calculated correction distortion coefficient vectors as its rows, and $\mathbf{q}_{min}$ and $\mathbf{q}_{max}$. Also passes on $\mathbf{C}^{cor}_{2D}$.

1. Use the *getCoeffs.m* program inputting $\mathbf{C}^{data\ 2D}$ and $\mathbf{C}^{exp\ 2D}$ to get the coefficient matrix S that stores distortion coefficients calculated in *getCoeffs.m*.
2. Calculate $\mathbf{q}_{min}$ and $\mathbf{q}_{max}$, the vectors containing the minimum and maximum x, y, z coordinates found in our dataset **C**.
3. For each position vector in $\mathbf{C}^{data\ 2D}$, calculate its corrected value by inputting it into the *correctDistortion.m* program alongside **S**, and $\mathbf{q}_{max}$ and $\mathbf{q}_{min}$, the boundaries of the box we scale the vectors into. Place these inside the 2D matrix $\mathbf{C}^{cor\ 2D}$ which we will print into our output file.
4. Output the calculated **S** and $\mathbf{C}^{cor\ 2D}$, as well as $\mathbf{q}_{max}$ and $\mathbf{q}_{min}$.

Calculating Distortion Correction Coefficients

- **File Name:** get_Coeffs.m
- **Input:** Matrices **P** and **Q**, **P** containing the correct (expected) coordinates and Q containing actual measurement coordinates as rows.
- **Output:** The 2D matrix **S** that contains the polynomial coefficients. This matrix is defined as the **C** matrix in the lecture as seen below [4], we called it **S** to avoid confusion with the **C** matrix containing the EM marker position for pivot calibration.

$$\begin{bmatrix} & \vdots & \\ F_{000}(\vec{\mathbf{u}}_s) & \cdots & F_{555}(\vec{\mathbf{u}}_s) \\ & \vdots & \end{bmatrix} \begin{bmatrix} c^x_{000} & c^y_{000} & c^z_{000} \\ \vdots & \vdots & \vdots \\ c^x_{555} & c^y_{555} & c^z_{555} \end{bmatrix} \cong \begin{bmatrix} & \vdots & \\ p^x_s & p^y_s & p^z_s \\ & \vdots & \end{bmatrix}$$

1. From **Q**, calculate $\mathbf{q_{min}}$ and $\mathbf{q_{max}}$ (could also be inputted into the function) from the maximum and minimum x, y, and z values found in **Q**.
2. Scale every **q** (every row of **Q**) and every **p** (every row of **P**) to box to be used with Bernstein polynomials, using *ScaleToBox.m.* Store these scaled **p** values in $\mathbf{P_{normalized.}}$
3. Calculate the Bernstein polynomials B using *bern.m* for each **q** to calculate $F_{i,j,k}$ for each $\mathbf{u}_s$ and construct the **F** matrix.

$$F_{i,j,k} = B_{5,i}(u_x)B_{5,j}(u_y)B_{5,k}(u_z)$$

4. Calculate the matrix $\mathbf{S = F^{-1}P_{normalized}}$.

Correcting Distortion Using the Coefficients

- **File Name:** correctDistortion.m
- **Input:** The 3D vector **q** that is to be corrected, $\mathbf{q_{max}}$ and $\mathbf{q_{min}}$ as (will be) calculated in Problem 2, and the matrix **S**, containing the correction coefficients..
- **Output: p**, the dewarped (corrected) values of the measured vector **q.**

1. Use *ScaleToBox.m* to scale the input **q** into **u**.
2. For each i, j, k value, access the coefficient vector $\mathbf{c_{i,j,k}}$ from the matrix **S** (as the vectors **c** constitute the rows of matrix **S)**, and multiply it by the Bernstein polynomials as

$$\mathbf{c_{i,j,k}}B_{5,i}(u_x)B_{5,j}(u_y)B_{5,k}(u_z)$$

3. Sum the coefficients vector multiplied by the polynomials across all i,j,k to obtain **p**, the vector of corrected values of **q**, as

$$\mathbf{p} = \sum_{i=0}^{5}\sum_{j=0}^{5}\sum_{k=0}^{5} \mathbf{c_{i,j,k}}B_{5,i}(u_x)B_{5,j}(u_y)B_{5,k}(u_z)$$

Calculating Bernstein Polynomials

- **File Name:** bern.m
- **Input:** The order of the Bernstein polynomial N (which is 5 in our case), the value k = 0, 1, 2, 3, 4, 5 that indicates which coefficient is being calculated, and a scalar v ∈ [0,1].
- **Output:** The Bernstein basis polynomial $B_{N,k}(v)$ defined as [4]:

$$B_{N,k}(v) = \begin{pmatrix} N \\ k \end{pmatrix} (1-v)^{N-k} v^k$$

Simply compute the value of the basis polynomial using the equation above.