Narrative Report

An overview of the structure of the computer program

Overall goal of the computer program

The structure of the program is built around a central `main.m` function which runs through a series of subroutines upon its execution. Its purpose is to:

(1) import the specified sensor and calibration values from the respective `.txt` and `.sur` files that live in the `INPUT` directory adjacent to `PROGRAMS`, `OUTPUT`, and `Debug`
(2) run the data through a series of processing functions
(3) plot the magnitude difference between s_k and c_k after each iteration of the ICP algorithm
(4) output a text file that contains
    - mode weights determined for the deformed atlas
    - position of sample points in the coordinates of the bone (`s`)
    - the points on the surface mesh that are closest to s_k = `F_reg * d_k (c)`
    - and the magnitude difference between s_k and c_k

Starting the program

When `main.m` is called, the command window displays the statement: `"Enter a letter from 'a' to 'k' (except i) to specify a dataset:"`. Once the user identifies the letter (and therefore the data set in the relative path to be read), the proper file names for

(1) marker positions in rigid body A,
(2) marker positions in rigid body B,
(3) body surface definition,
(4) the deformed atlas mode file, and
(5) sample readings of optical tracker

are read into their respective sets (i.e. *a_set*, *b_set*, etc).

Verify that vertices of the mesh corresponds to "Mode 0"

This step validates the mesh and atlas files by ensuring that the vertices in the mesh file are the same as the vertex displacements for mode 0, which is the average shape of the bone.

Computing F_A and F_B

We compute *F_A* and *F_B* by calling the `find_frame_transformation()` function, which takes a vector set (e.g. *a_set*) and a two-dimensional array of vectors formed by positions of markers in each sample reading (e.g. `A_sample`). This function is a pointer script that passes the first parameter and each set of vectors in the second parameter into the function defined in `registration3dTo3d.m`, which is a "3D point set to 3D point set registration algorithm" that computes a frame transformation such that $a = F^{-1}b$, where *a* and *b* are the first and second parameter of `registration3dTo3d()` respectively.

Computing *d*

Access to *F_A* and *F_B* then allows us to compute $d = F\_B^{-1} * F\_A * a\_tip$, which is the set of vectors representing the position of pointed tip A in the local coordinates of rigid body B.

The ICP algorithm

Initialization

The first step of the ICP algorithm is Initialization, where all the arrays that will be used in the while loop are initialized. *F_reg* is also initialized to the Identity transformation, which is our initial guess such that $c_k = F\_reg * d_k$.

Model matching iteration

With an initial approximation of F_reg, we find the closest point on the mesh for each datapoint, then find the coordinates of the points we found in the deformed atlas. We then obtain the mode weights by solving a least squares problem. We iterate the aforementioned steps until the mode weights converge.

Transformation update

The next step in the ICP while loop is to update `F_reg` using our `registration3dTo3d()` function to determine the frame transformation from `A` to `B`. The values for `sigma`, `epsilon_max`, and `epsilon_avg` of the current iteration are also calculated according to their respective equations.

Adjustment

The current match distance threshold for determining whether a data point is an outlier is set to the 95th percentile of the magnitude difference between d_k and c_k in the current iteration.

Termination test

This test is used as the condition for terminating the while loop: if σ, ε_{max}, or ε_{avg} are less than the desired thresholds, and $0.95 \leq \varepsilon_{avg(n)}/\varepsilon_{avg(n-1)} \leq 1$ for more than 10 iterations, the while loop will be terminated. If this condition cannot be satisfied after 75 iterations, the while loop will be terminated as well.

Output

Finally, the computed mode weights, s_k, c_k, and magnitude difference between s_k and c_k are then written out to `NAME-Output.txt` in the `OUTPUT` directory. This program also outputs a plot illustrating the convergence of s_k and c_k.

You may see the appendix for a list of variables defined in our program and used in our description above.

The algorithmic steps followed

The Model Matching step in the ICP algorithm

1) Matching

We first compute s_k = F_reg * d_k, which are the position of sample points in the coordinates of the bone, then use the function `find_closest_point_to_mesh()`, which makes use of the function `find_closest_point_to_triangle()`, to determine c_k, the points on the surface mesh that are closest to s_k. By comparing the threshold with the magnitude difference between the calculated and actual position of c_k, we determine whether the point is an outlier in the set of data points. If not, d_k and c_k are appended to `A` and `B` respectively, and their magnitude difference is appended to `E`. This step also saves the indices of the triangle that c_k is located on.

2) Converting c_k to coordinates in the deformed atlas

The next step in the model-matching while loop is to compute the coordinates of the vertices of the triangle that each c_k is on in the deformed atlas. We do so using the `get_deformed_coord.m` function, which add the coordinates of the given point in the "average shape" atlas and the corresponding vector displacements of each mode of that point multiplied by the current estimate of the mode weights:

$$\vec{\mathbf{m}}_s = \vec{\mathbf{m}}_{0,s} + \sum_{m=1}^{Nmodes} \lambda_m{}^{(t)} \vec{\mathbf{m}}_{m,s}$$
$$\vec{\mathbf{m}}_t = \vec{\mathbf{m}}_{0,t} + \sum_{m=1}^{Nmodes} \lambda_m{}^{(t)} \vec{\mathbf{m}}_{m,t}$$
$$\vec{\mathbf{m}}_u = \vec{\mathbf{m}}_{0,u} + \sum_{m=1}^{Nmodes} \lambda_m{}^{(t)} \vec{\mathbf{m}}_{m,u}$$

Then, we find the barycentric coordinates of c_k with respect to the vertices of the triangle in the deformed atlas, from which we could compute the coordinates of c_k in the deformed atlas:

$$\vec{\mathbf{c}}_k^{(t)} = \zeta_k \vec{\mathbf{m}}_s + \xi_k \vec{\mathbf{m}}_t + \psi_k \vec{\mathbf{m}}_u$$

As we could express c_k in terms of mode coordinates:

$$\vec{\mathbf{c}}_k{}^{(t)} = \vec{\mathbf{q}}_{0,k} + \sum_{m=1}^{N_{modes}} \lambda_m{}^{(t)} \vec{\mathbf{q}}_{m,k}$$

where

$$\vec{\mathbf{q}}_{m,k} = \zeta_k \vec{\mathbf{m}}_{m,s} + \xi_k \vec{\mathbf{m}}_{m,t} + \psi_k \vec{\mathbf{m}}_{m,u}$$

we can obtain the mode weights by solving a least squares problem Ax = B, where

$$A = \begin{bmatrix} & & . & \\ & & . & \\ & & . & \\ q_{1,k} & q_{2,k} & \cdots & q_{n,k} \\ & & . & \\ & & . & \\ & & . & \end{bmatrix} \quad x = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ . \\ . \\ . \\ . \\ \lambda_n \end{bmatrix} \quad B = \begin{bmatrix} . \\ . \\ . \\ c_k - q_{0,k} \\ . \\ . \\ . \end{bmatrix}$$

3) Adjustments

We then update the vertices on the mesh using the newly computed mode weights.

4) Termination

To determine when to terminate the model-matching while loop, we ensure that the difference between the mode weights calculated in the previous iteration is similar enough compared to that obtained in the current iteration (their ratio should be between 0.95 and 1 or that their differences have to be smaller than 0.1) for 5 iterations consecutively. The model matching iteration will terminate after 30 iterations if neither of these conditions have been met.

Please refer to the report from PA4 for descriptions of:

- ICP algorithm (except the matching step)

and PA3 for descriptions of:

- `find_closest_point_to_triangle` algorithm

and PA1 for descriptions of:

- Registration algorithm
- Averaging vector objects

The mathematical approach taken

The corresponding coordinates of a point in the deformed mesh

We find the coordinates of `v` in the deformed mesh using the formula:

$$\vec{v} = \vec{v}_0 + \sum \lambda_m \vec{v}_m$$

We add its coordinates in the atlas of "average shape" to the sum of the product of each mode weights with each mode of the vertex, which will give the corresponding coordinates in the deformed mesh.

We use this formula to compute the coordinates of the vertices m_s, m_u, m_t of the triangle that a point c_k is on, as well as the coordinates of c_k.

Computing barycentric coordinates of a point

To find the barycentric coordinates of a point given the vertices of the triangle that the point is on {`s`, `t`, `u`}, i.e. solving for λ, μ, υ in

$$p = \lambda s + \mu t + \upsilon u$$

Because

$$\lambda + \mu + \upsilon = 1$$

We could formulate the equation above as

$$p = (1 - \mu - \upsilon)s + \mu t + \upsilon u$$
$$p = s + \mu t - \mu s + \upsilon u - \upsilon s$$
$$p - s = \mu(t - s) + \upsilon(u - s)$$

which is equivalent to expressing vectors `st`, `su`, `sp` in the coordinate system of `s`. If we let

$$v_0 = t - s, v_1 = u - s, v_2 = p - s$$

We can rewrite the equation as

$$v_2 = \mu v_0 + \upsilon v_1$$

Then we can formulate a system of linear equation to solve for μ, υ by taking the dot product of both sides with v_0 and v_1:

$$\mu(v_0 \cdot v_0) + \upsilon(v_1 \cdot v_0) = v_0 \cdot v_2$$
$$\mu(v_0 \cdot v_1) + \upsilon(v_1 \cdot v_1) = v_1 \cdot v_2$$

Computing system of linear equation

Now we solve this system using Cramer's rule which states that for a system Ax = b,

$$x_i = \frac{det(A_i)}{det(A)}$$

where A_i is the matrix formed by replacing the i^{th} column of A by the column vector b. In our case,

$$A = \begin{bmatrix} v_0 \cdot v_0 & v_1 \cdot v_0 \\ v_0 \cdot v_1 & v_1 \cdot v_1 \end{bmatrix}, x = \begin{bmatrix} \mu \\ v \end{bmatrix}, b = \begin{bmatrix} v_2 \cdot v_0 \\ v_2 \cdot v_1 \end{bmatrix}$$

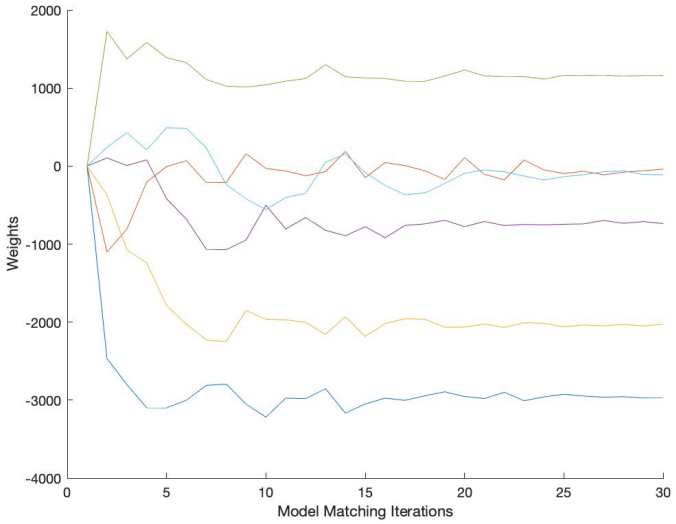
Determination of the termination conditions of the model matching iteration

To ensure that the mode weights have converged, we compare the difference between the mode weights calculated in the previous iteration to that obtained in the current iteration, and check if their ratio is between 0.95 and 1, i.e.

$$0.95 > \frac{|\wedge^{t+1} - \wedge^{t}|}{|\wedge^{t} - \wedge^{t-1}|} > 1$$

or if their differences are smaller than 0.1.

In addition, we have run the model matching step for 100 iterations using F_reg = I as the initial guess, and we find that the mode weights converge after 30 iterations for most datasets. The plot below shows how the mode weights converge for dataset b:



Therefore, we have decided to terminate the model-matching while loop after 30 iterations if none of the aforementioned conditions have been met.

Determination of the termination conditions of the ICP algorithm

Our goal was to repeat the process of statistical analysis on the magnitude difference between s_k and c_k after each iteration of the ICP algorithm throughout 150 iterations. We were unable to do this because our tree search is prohibitively slow (despite its accuracy). Since we were unable to improve the balance of the tree, it would take days to run the number of iterations that we originally intended. Instead, we ran one full iteration in order to generate some results for analysis. To fix this efficiency problem, one could simply substitute a faster tree search algorithm into the program and the ICP algorithm should work quickly and effectively. Please refer to our PA4 report for a description of the "Determination of Termination Conditions" procedure which we would have followed here if our tree algorithm were sufficiently fast to determine the three items below:

Threshold for the magnitude difference between s_k and c_k in the next iteration

Thresholds for σ, ε_{max}, and ε_{avg} in the termination test

Number of iterations required to meet termination conditions

Please refer to the report from PA1 for descriptions of:

- Data structures for Vector, Rotation, and Frame

and please refer to PA3 for descriptions of:

- Data structure for Node and Kd Tree
- The Mathematical Principles behind searching the closest point on a triangle

The steps taken to verify that the program is working correctly

Unit Testing

We have implemented a series of unit tests in the file `/DEBUG/unit_tests.m` (which includes those for functions and classes also in PA4) to test our newly added functions:

- `get_barycentric_coord()`,
- `get_deformed_coordinates()`, and
- `dot()`, a method of the `Vector` class.

We tested these functions with edge cases and they work as expected.

Comparing our output data against the given answers

We've written `discrepancy.m` in the DEBUG folder to compute the discrepancy of the mode weights and the of x, y, z coordinates of each s_k and c_k between

(1) the output file in the OUTPUT directory and
(2) the answer given in the INPUT directory.

This program outputs and saves two bar charts illustrating the discrepancies (plots would have been included in the next section).

Please refer to the report from PA4 for descriptions of:

- The magnitude difference between s_k and c_k in our output

A tabular summary of the results obtained for unknown data

The magnitude difference between s_k and c_k in our output

If the algorithm were able to run sufficiently fast, we would have instructed the reader to refer to the `.txt` files in the `OUTPUT` directory to view the data described by plots that would have been in this section, which illustrate the magnitude difference between s_k and c_k after each iteration of the ICP algorithm.

Discrepancies between our output and the given answers

We would have also produced four plots for each dataset which describe the millimeter-scale discrepancy between our output data and the given answers from the `INPUT` directory. The horizontal line in our plots would have shown the average discrepancy across all points. For the plot titled “Comparing discrepancies in s and c”, we would have combined the error in `s` and `c` to look for errors that occur in both. Black bars would have represented errors that are present in both `s` and `c`, magenta bars would have represented errors that were only found in `s`, and cyan bars would be errors that are only found in c. We would have also produced plots to illustrate the discrepancies in the mode weights.

A short discussion of results from running your program

The magnitude difference between s_k and c_k in our output

By plotting the average magnitude differences over many iterations, we wish to confirm that our algorithm is acting in accordance with our desired goal. We hope to see that the magnitude difference (mm) decreases exponentially and then eventually finds stability at levels closer to 0. Each particular level would depend on the dataset being processed, but all of the plots should show very similar shapes of `epsilon_avg`'s exponential decay over time.

Discrepancies between our output and the given answers

We hope to see minimal discrepancy between outputs (optimally below 0.1mm), which would confirm that our algorithm is quite accurate.

A short statement of who did what

After Vivian added code to `main.m` from PA4 for I/O, John wrote the majority of the code for model matching and discussed mathematical approaches with Vivian along the way. Vivian helped with debugging the code by comparing our output with the given answers while John wrote the unit tests for the new functions and methods. Vivian wrote the majority of the programming listing, and Vivian and John split responsibilities of the report evenly.

Appendix

From BodyA.txt

a_set: a set of vector *a* representing coordinates of marker LEDs in body coordinates (1 row)
a_tip: a vector representing coordinates of the tip in body coordinates

From BodyB.txt

b_set: a set of vector *b* representing coordinates of marker LEDs in body coordinates (1 row)
b_tip: a vector representing coordinates of the tip in body coordinates

From MeshFile.sur

v_set: a set of vectors representing coordinates of vertices in CT coordinates (1 row)
triangle_v_set: vertex indices of the three vertices for each triangle (1 row per triangle)

From Mode.txt

m: vector displacements

From SampleReadings.txt

A_sample: sets of vector A representing coordinates of A body LED markers in tracker coordinates (1 sample per row)
B_sample: sets of vector A representing coordinates of B body LED markers in tracker coordinates (1 sample per row)
num_modes: number of modes needed for this dataset

Derived variables

F_A: a set of frames such that *F_A * a = A* (1 row)
F_B: a set of frames such that *F_B * b = B* (1 row)
d_set: a set of vector *d* such that $d = F\_B^{-1} * F\_A * a\_tip$ (1 row)
N: number of sample frames
A: an array holding d_k that is not an outlier in the current iteration
B: an array holding c_k if its corresponding d_k is not an outlier in the current iteration
n: iteration number
count: the number of times that the termination conditions have been satisfied
F_reg: the transformation from c to d
threshold: the threshold for the magnitude difference between s_k and c_k beyond which the d_k is considered an outlier

E: an array holding the magnitude difference between s_k and c_k for the current iteration
all_E: a master list of magnitude differences between s_k and c_k
sigma: the variance of the c_k from s_k in the current iteration
epsilon_max: maximum magnitude difference between c_k from s_k in current iteration
epsilon_avg: average magnitude difference between c_k from s_k in current iteration
s_set_curr: set of vector *s* such that *s = F_reg * d* in current iteration
c_set_curr: set of vector *c* which are points on the surface mesh that are closest to s_k in the current iteration
s_set__prev: previous iteration's set of vectors *s* such that *s = F_reg * d*
c_set__prev: previous iteration's set of vectors *c* which are points on the surface mesh that are closest to s_k