

# CIS Software Manual

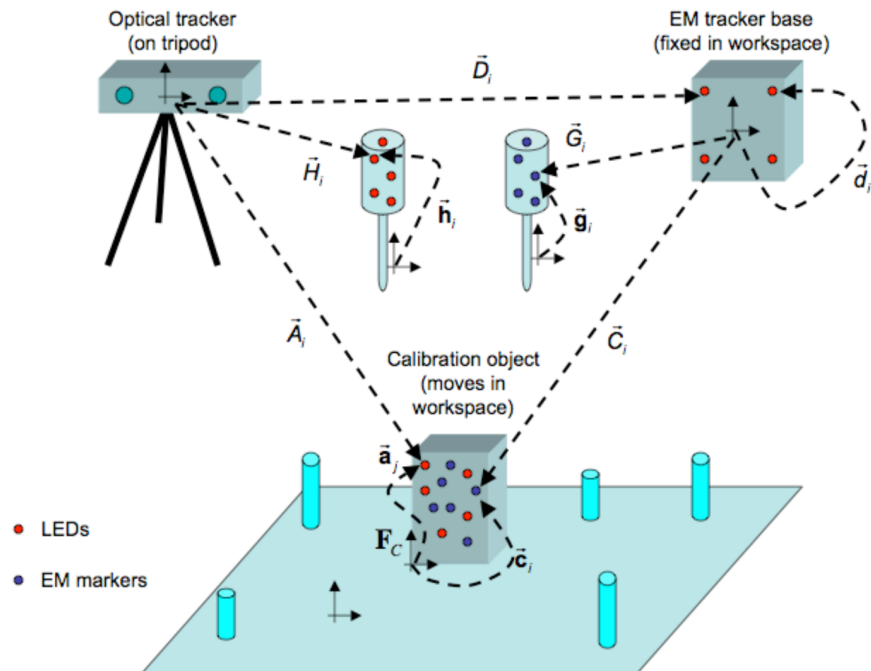
Release v1.0

Joshua Shubert and Nathan Smith

Oct 25, 2016

## Contents

1. Introduction
  - 1.1 PA1
2. Mathematical Approach
  - 2.1 Point Cloud Registration
  - 2.2 Pivot Calibration
3. Algorithmic Approach
  - 3.1 Parsing
  - 3.2 Point Cloud Registration
  - 3.3 Pivot Calibration
4. Structure of Program
  - 4.1 Function List
  - 4.2 File List
5. Results and Discussion
6. Additional Information



## 2 Mathematical Approach

### Point Cloud Registration

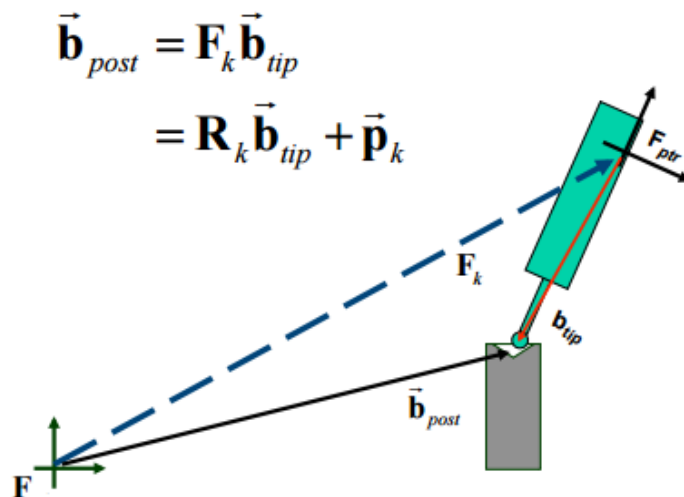
A number of least squares methods could be used to determine a transformation matrix for a 3D point set registration. We chose a solution from 'A Method for Registration of 3-D Shapes', by Besl and McKay, 1992. We seek to solve for  $R$  and  $t$ , in the equation  $B = R*A + t$ , where  $(R, t)$  are transforms applied to the dataset  $A$  to align it with dataset  $B$ , as closely as possible. Finding the optimal rigid transformation matrix can be broken down into the following steps:

1. Find the centroids of both datasets
2. Bring both matrices to the origin then find the optimal rotation,  $R$
3. Find the translation  $t$

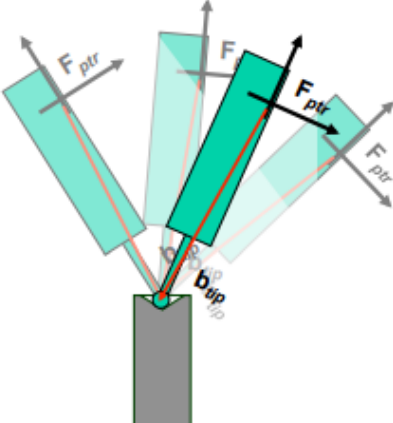
The centroids are simply found by calculating the average  $(x,y,z)$  values for each point in datasets  $A$  and  $B$ . To find the optimal rotation, we make use of singular value decomposition. SVD decomposes the matrix into three other matrices, such that  $[U,S,V] = \text{SVD}(E)$ , and  $E = USV^t$ . We first re-center both datasets so that both centroids are at the origin, removing the translation component. We then accumulate matrix  $H$ , and solve for  $[U,S,V] = \text{SVD}(H)$ , and find the rotation matrix  $R = VU^t$ . The translation is  $t = -R * \text{centroid}(A) + \text{centroid}(B)$ .

### Pivot Calibration

To perform pivot calibration, we need to simultaneously determine the location of the fiducial post  $b_{\text{post}}$  and the position of the tip of the tracked tool  $b_{\text{tip}}$ . We use a least squares method to solve for these values simultaneously. This method requires the transformation,  $F_k$  between the tracker and the tool for each of the  $k$  frames of data, which we determine using our point cloud registration technique between the tracker and the fiducials placed on the tool.



To determine the local tool frame, we first compute the midpoint of all the fiducial locations,  $G_i$ , in the tracker frame. This gives us a reference,  $g_0$ , from which to localize all the other fiducial locations. Once all the  $g_i$  are localized we compute the transformation between the  $g_i$  and the  $G_i$  for each frame  $k$  of data to get a set of  $F_k$ . We then solve the below least squares problem to solve for  $b_{post}$  in the tool frame and  $b_{tip}$  in the tracker frame.

$$\begin{bmatrix} \vdots & \vdots & \vdots \\ \mathbf{R}_k & -\mathbf{I} \\ \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} \mathbf{b}_{tip} \\ \mathbf{b}_{post} \end{bmatrix} \cong \begin{bmatrix} \vdots \\ -\mathbf{p}_k \\ \vdots \end{bmatrix}$$


### 3 Algorithmic Approach

#### Parsing

Our program was developed in Python. The Numpy library was used as a Cartesian math package for 3D points, rotations, and frame transformations. The linear algebra package numpy.linalg was also used for matrix operations including determinants and inversions. The first step was to develop a parser that could interpret and load the provided data. The parser for each file observes the first line of the file to determine the number of elements and frames for the dataset. It then iterated through the dataset line by line, accumulating arrays where each element represents a 3-dimensional point. These arrays are stored as numpy arrays, allowing for numpy functions and transformations to be applied.

#### Point Cloud Transformation

We next developed the algorithm for point cloud to point cloud registration, and determining the rotation and translation components of the transformation between them. As covered above, the function uses singular value decomposition to separate the  $R$  and  $t$  elements of the transformation. We first load in both point clouds, represented as numpy arrays. The centroids of each array are determined by finding the mean of the  $[x,y,z]$  values of each element in the arrays. These mean values are then subtracted from every point in each array to localize the arrays and determine the rotation component. Numpy's linalg.svd function is used to decompose  $H$  into  $U$ ,  $S$ ,  $V$ , and  $R$  is then calculated as the multiplication of  $V$  and  $U$ . The

translation  $t$  is then calculated as  $-R * \text{centroid}(A) + \text{centroid}(B)$ , using the `numpy.dot` function to calculate matrix dot products. These  $R$  and  $t$  values are then rounded and returned by the function.

## Pivot Calibration

Both the EM pivot calibration and the Optical pivot calibration are solved using the same function `calibrate_pivot()`. For the optical pivot calibration there is a wrapper around `calibrate_pivot()` called `opt2em_calibrate_pivot()` that translates the optical fiducial locations from the optical tracker frame to the EM tracker frame before performing the pivot calibration. In essence, `calibrate_pivot()` performs the same tasks as mentioned above for pivot calibration. A set of points describing the locations of the fiducial trackers on the tool in the tracker frame are localized by first computing the midpoint of the set of midpoints using `compute_midpoint()`, and then creating a new set of points where each point is the original tracker frame point with the location of the midpoint subtracted from it using `localize()`. Then point cloud to point cloud registration is performed between these two sets of points to get the transformation from tracker to tool for this frame of data.

Once the frame transformation has been calculated for each frame of data, the frames are formatted into a matrix according to the following graphic by the `pivot_calibration()` function:

$$\begin{bmatrix} \vdots & \vdots \\ \mathbf{R}_k & -\mathbf{I} \\ \vdots & \vdots \end{bmatrix} \begin{bmatrix} \vec{\mathbf{b}}_{tip} \\ \vec{\mathbf{b}}_{post} \end{bmatrix} \equiv \begin{bmatrix} \vdots \\ -\vec{\mathbf{p}}_k \\ \vdots \end{bmatrix}$$

Once the  $R$  and  $p$  matrices are assembled, we call Numpy's `lstsq()` function to solve the least squares problem. It returns a vector  $b$  that contains the  $b_{tip}$  and  $b_{post}$  solutions.

## 4 Structure of Program

The functions making up the assignment 1 implementation are as follows:

p1_driver	The main function of the program. Loads in each of the datasets, performs pivot calibration and point cloud transformations. Creates and populates output files for evaluation.
read_cal_readings	Reads the cal_readings file into a numpy array dataset.
read_cal_body	Reads the cal_body file into a numpy array dataset.
read_em_pivot	Reads the em_pivot file in a numpy array dataset.
read_opt_pivot	Reads the opt_pivot file into a numpy array dataset.
get_transform	Takes in two matrices A and B. Determines the rotation matrix R and translation component t such that $B = R \cdot A + t$ , and returns R and t.
calibrate_pivot	Takes in the em_pivot dataset G, the set of points where the tool is in contact with a fiducial pin. Returns p_tip, the vector from the tool frame to the end of the tool, and p_post, the vector from the EM base frame to the top of the fiducial post.
compute_midpoint	Called by calibrate_pivot, returning the mean points [x, y, z] of the input array.
localize	Called by calibrate_pivot, returning a numpy array with the mean points [x,y,z] subtracted from each point element.
opt2em_calibrate_pivot	Calculates the transform between the EM and Optical tracker for each frame of data.
p1_tests	Runs unit tests on the dataset loading, transformation, and pivot calibration functions to ensure they're working properly outside of the final output.

These functions have been organized into the following files:

p1_driver.py	The main program driver, producing output files.
p1_tests.py	Contains the unit tests for all core functions of the program.
read_datasets.py	Contains all dataset parsing functions.
transformation.py	Contains all point cloud transformation functions.

<b>calibrate_pivot.py</b>	Contains all pivot calibration functions.
---------------------------	---

Additionally, data and output folders contain all input datasets, and all output produced for each of these datasets, respectively. To run the program on all datasets at once, navigate to the source folder, and enter:

**python p1\_driver.py**

Then, refer to the output folder to find all generated output files. Each output1 file contains the em and opt pivot calibration position estimates as the first two lines. The remaining lines cover the expected values C for each point and frame in the respective dataset. Additionally, each output-difference file lists the difference between the expected and true C values for each of the datasets.

Extensive comments have been provided in each of the source files to outline the structure and flow of the program. The driver iterates through each of the 11 provided datasets, each time setting the names of each of the dataset components and loading them into the system as a numpy array. The EM and Optical pivot calibration functions are then called, with their values stored to be printed in the output file. The C values are then iterated through frame by frame, with the expected Cs calculate through get\_transform and appended to the final expected C list. Finally, output files are generated and saved in the output folder.

To run the program's unit tests, navigate to the source folder and enter:

**python p1\_tests.py**

All unit tests have been found to pass properly, as outlined in the next section.

## **External Libraries**

As stated above, the NumPy library was used as our Cartesian math package. The specific submodules of NumPy used were np.linalg, np.matrix, and np.testing.

## 5 Results and Discussion

### Validation

We used several approaches to ensure the validation and accuracy of our program. Methods were developed incrementally, first verifying that dataset input and matrix generation was working as intended, before moving on to point cloud transformations and ultimately pivot calibration. To assess the accuracy of our final output, we generate not only an output1.txt file for each dataset, but also an output-difference.txt for each file to determine the accuracy of our C expectations. Pivot calibration output was also compared against the auxiliary files for each of the debug datasets. In addition to assessing final output, unit tests were written to ensure the accuracy of the program's subroutines. These include the dataset input, determining the rotation and translation matrices during point cloud registration, producing expected values from given transformations, and validating the pivot calibration output.

### Point Cloud Registration

We have been able to ensure that point cloud registration is working properly by finding the transformation of one point cloud to another and then the opposite. Multiplying these two transformation matrices together resulted in an identity matrix, as was expected. We also tested the input data set, ensuring that our final output was within a reasonable tolerance range.

### Pivot Calibration

The position of the tip of the probe when calibrated by RM gave results very similar to those listed in the auxiliary files for each dataset, well within our tolerance range. For further information on testing, please refer to the p1\_tests.py file in the source folder. To view our output for each dataset, refer to the output folder.

### Unknown Data Summary

The following table covers the em and opt pivot estimations, and average difference between the expected and listed C points, for each of the unknown datasets:

Dataset	EM Pivot	Optical Pivot	C Difference
unknown-h	[188.238, 231.994, 215.170]	[398.694, 393.650, 203.851]	20.525
unknown-i	[190.662, 204.362, 210.669]	[396.570, 398.576, 204.645]	17.336
unknown-j	[192.482, 194.217, 200.665]	[390.077, 397.445, 195.166]	16.636
unknown-k	[201.905, 187.854, 197.049]	[398.649, 412.293, 197.833]	16.509



For the debugging datasets where an auxiliary file was provided we were able to compare our EM and Optical pivot outputs against the provided coordinates. It was found that our output coordinates were very accurate, as shown below:

Dataset	EM Pivot	Optical Pivot
debug-a-out	[191.74, 200.48, 198.22]	[401.57, 392.03, 191.57]
debug-a-aux	[191.72, 200.48, 198,22]	[401.60, 392.03, 191.59]

Results were found to be similarly accurate for each of the output datasets. For further information, please refer to the driver's output in the output/ folder.

## 6 Additional Information

This program was developed collaboratively by Josh Shubert and Nathan Smith, with the vast majority of the program written together. Nathan covered most of the file input and output, while Josh covered most of the pivot calibration functions, however all files were developed with input from both members.