

---

# CIS Software Manual

*Release b2014.10.28 (15:17 UTC)*

**Andrew Hundt and Alex Strickland**

October 29, 2014

## Contents

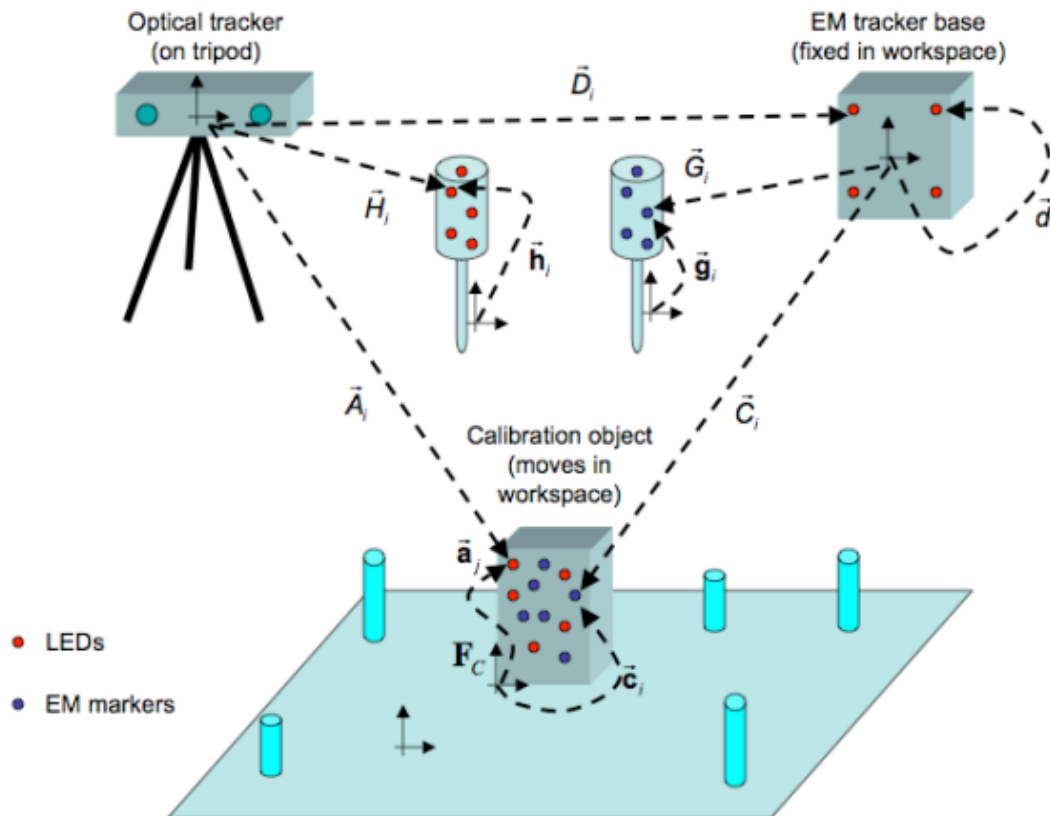
<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Mathematical Approach</b>	<b>3</b>
2.1	Point Cloud Registration . . . . .	3
2.2	Pivot Calibration . . . . .	4
<b>3</b>	<b>Algorithmic Approach:</b>	<b>4</b>
3.1	Parsing . . . . .	4
3.2	Transforms . . . . .	4
3.3	Pivot Calibration . . . . .	4
<b>4</b>	<b>Structure of the Program</b>	<b>5</b>
4.1	Important Functions and Descriptions . . . . .	5
<b>5</b>	<b>Results and Discussion</b>	<b>6</b>
5.1	Validation . . . . .	6
5.2	Point Cloud Registration . . . . .	6
5.3	Calibration . . . . .	6
<b>6</b>	<b>Status of results</b>	<b>6</b>
<b>7</b>	<b>Additional Information</b>	<b>7</b>
7.1	Features . . . . .	7
7.2	People . . . . .	7
	Software Development . . . . .	7
7.3	Quick Start . . . . .	7
	First Steps . . . . .	7
	Advanced Information . . . . .	9
7.4	Read the Data format . . . . .	10
7.5	Getting Help . . . . .	10
7.6	Reference . . . . .	11
	Source Package . . . . .	11
7.7	Download . . . . .	11
	Source Code . . . . .	11
	Documentation . . . . .	11
7.8	Installation . . . . .	12
	Prerequisites . . . . .	12

Configure . . . . .	12
Build . . . . .	12
Test . . . . .	12
Install . . . . .	13

---

## 1 Introduction

The purpose of the assignment was to develop an algorithm for a 3D point set to 3D point set registration and a pivot calibration. The problem involved a stereotactic navigation system and an electromagnetic positional tracking device. Tracking markers were placed on objects so the optical tracking device and an electromagnetic tracking device could measure the 3D positions of objects in space relative to measuring base units. These objects were then registered so that they could be related in the same coordinate frames. Pivot calibration posts were placed in the system so pivot calibration could be performed and the 3D position of two different probes could be tracked throughout the system. The diagram below from the assignment document gives a visual description of the system.



## 2 Mathematical Approach

### 2.1 Point Cloud Registration

A number of least squares methods could be used to determine a transformation matrix for a 3D point set registration. We selected to use Horn's method because a rotation matrix is always found and no iterative approximation is involved. The first step is to find the centroid of the point clouds in the two different coordinate systems. Then the centroid is subtracted from each point measurement of the separate point clouds so the points will be relative to the centroid. Next an H matrix is created which is the sum of the products of each corresponding point in the two frames. A real symmetric G matrix is then created from the sums and differences of the elements in the H matrix which was previously created.

Next, the eigenvalues and corresponding eigenvectors of the G matrix were calculated. The eigenvector corresponding to the most positive eigenvalue represents the unit quaternion of the matrix. Once the quaternion is known, the rotation matrix can be found using Rodriguez's formula. The translation between the two coordinate systems is next found from the difference between the centroid of the known point cloud and the scaled centroid of the unknown point cloud. Finally a homogeneous transformation matrix could be made to know the frame transformation between the two point clouds.

## 2.2 Pivot Calibration

For the pivot calibration, singular value decomposition was used to estimate the orientation of the probe by finding the positions of the centroid of the tracked markers on the probe and the tip of the probe. First, a matrix was created which consisted of the rotation matrices calculated in each frame and the negative identity matrix found using Horn's method.[1] Next, a vector was created which consisted of the stack of the translation vectors in each frame also found using Horn's method. The singular value decomposition of the matrix was performed to split the matrix into the matrices containing the singular values, the left-singular vectors, and the right singular vectors. Once this was done, the vector between the centroid of the tracked markers on the probe and the probe tip could be approximated using the SVD matrices and the translation vectors of each frame.

- [1] Horn, Closed-form solution of absolute orientation using unit quaternions, Optical Society of America (1987)

## 3 Algorithmic Approach:

### 3.1 Parsing

We developed our algorithm using C++. The Eigen library was used as a Cartesian math package for 3D points, rotations, and frame transformations. The Boost library was also used to write a parser file and develop various aspects of our algorithms. The first step was to write parser code that could interpret the given data. The parser needed to interpret which data set was being entered, the number of frames in each data set, and which markers were being tracked in the data set. The parser would store the data as Eigen matrices to be easily used for our algorithms. A diagram below shows how the parser function interpreted and stored the data.

### 3.2 Transforms

Once the data was parsed, two matrices containing marker positions in different coordinate frames was put in the function `hornRegistration` to determine the corresponding transformation matrix between the two frames. The first step of the `hornRegistration` was to find the two centroids of two 3D marker positions and subtract it from each marker position using functions in the Eigen library. The next step was to put these values in a function that would create a 3x3 H matrix. Once this was done, the H matrix could be put in a separate function that would calculate the 4x4 G matrix. The eigenvalues and the corresponding eigenvectors of the G matrix were next calculated by using functions of the Eigen library. A vector of each eigenvalue and the corresponding eigenvector was then created so that the eigenvalues could be sorted to find the most positive eigenvalue and its corresponding eigenvector which represented the unit quaternion of the rotation. Next, the 3x3 rotation matrix was created by an Eigen function that converted a unit quaternion into the corresponding rotation matrix. Finally, the translation vector between the two centroids was calculated and a 4x4 homogeneous transformation matrix was created by using another function that takes a rotation matrix and a translation vector and outputs the corresponding transformation matrix.

### 3.3 Pivot Calibration

Next a pivot calibration algorithm was created which used both the parser and `hornRegistration` algorithms mentioned above. First, the tracker data was parsed into separate matrices which corresponded to each frame of tracked data. Each

matrix of frame data was compared to the base matrix frame using the `hornRegistration` function described above and the corresponding homogeneous transformation from the base frame to the current frame was found. The rotational component of each frame was put into an Eigen matrix and the translational component of each frame was put into an Eigen vector with the form described in the mathematical approach above. The function of JacobiSVD of the Eigen library was then used to solve the least squares vector between the rotational matrix and translation vectors. The least squares vector contained approximated orientation of the probe and the position of the probe tip.

## 4 Structure of the Program

The most important files include:

File name	Description
<b>hornRegistration.hpp</b>	Functions for implementation Horn's method of Point Cloud to Point Cloud registration.
<b>PivotCalibration.hpp</b>	Functions for implementing Pivot Calibration.
<b>cisHW1test.cpp</b>	An extensive set of unit tests for the library.
<b>cisHW1main.cpp</b>	Main executable source, contains cmdline parsing code and produces output data.
<b>parseCSV...</b>	File parsing functions are in <b>parseCSV_CIS_pointCloud.hpp</b> .

The software is structured as a set of header only libraries in the include folder, which are utilized by the unit tests, main, and any external libraries that choose to use these utilities.

Each function includes substantial doxygen documentation explaining its purpose and usage. This documentation can be viewed inline with the source code, or via a generated html sphinx + doxygen website generated using CMake. Here is a list of the most important functions used in the program is a brief description of each of them.

### 4.1 Important Functions and Descriptions

#### **EigenMatrix()**

Computes the eigenvalues and corresponding eigenvectors from a given G matrix. It outputs a rotation matrix corresponding to the unit quaternion of the largest positive eigenvalue

#### **homogeneousmatrix()**

Creates a 4x4 homogeneous matrix from a derived rotational matrix and translational vector

#### **hornRegistration()**

Computes the homogeneous transformation matrix F given a set of two cloud points. It is comprised of the various functions listed above

#### **homogeneousInverse()**

Computes the inverse of a given homogeneous matrix

#### **registrationToFirstCloud()**

Parses the data and runs the `hornRegistration` function for pivot calibration

#### **transformToRandMinusIandPMatrices()**

Creates the A and b components of the form  $Ax=b$  for singular value decomposition. A is of the form  $[RI-I]$  while b is of the form  $[-p]$  where R is the stack of rotational matrices of the F transformation matrices, I is stack of 3x3 identity matrices, and p is the stack of the translational vectors of the F transformation matrices.

#### **SVDSolve()**

Computes the x of the least squares problem  $Ax=b$  using singular value decomposition when the stack of matrices in given

### **Hmatrix()**

Computes a sum of the products H matrix given a set of two cloud points

### **Gmatrix()**

Computes a sum of the differences of the given H matrix

### **pivotCalibration()**

Computes the pivot point position from tracking data using the SVDSolve(), registrationToFirstCloud(), and transformToRandMinusIandPMatrices() functions

## **5 Results and Discussion**

### **5.1 Validation**

We took several approaches to the validation of our software. These include manual and automatic execution of the supplied test data, the implementation of unit tests to verify the data, and initial integration of continuous integration software to catch errors early. We implemented a battery of unit tests to verify the basic functions and ensure they are running correctly.

### **5.2 Point Cloud Registration**

We have been able to ensure that point cloud to point cloud registration is working correctly 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 which would be expected. We tested the input data set as well, ensuring that we were within the given tolerance range. Our program produces nearly exact results when the data was run with no error. When error such as EM distortion, EM noise, and OT jiggle, were introduced in the data, our results were still very close to the expected results and were well within our tolerance range. This shows the strength of Horn's method and since it requires no special case exceptions for a solution, we concluded it was the best method of the one's taught in class.

### **5.3 Calibration**

The position of the tip of the probe when calibrate by EM also gave us results well with in out tolerance levels. Again, it could be seen that our results were less accurate when error was introduced, but not to an unreasonable degree. We did encounter a bug when trying to calibrate the optical probe in the EM coordinate system. Our results for the x and y values of the pivot point were in the tolerance range of the actual pivot point. However, we ran into a large systematic shift in the negative z-direction of our output data. We believe that there was a possible ordering or directional issue in our algorithmic transforms. Since the shift was systematic throughout all the test data, the value of the error was well know and could accounted for if this were a real system until the bug was found.

## **6 Status of results**

Andrew and Alex spent approximately equal time on the assignment, with significant amounts of time spent pair programming. Both contributed equally to the implementation and debugging of funcitons.

## 7 Additional Information

### 7.1 Features

#### Horn Registration

- Point cloud to point cloud transformations .

#### Pivot calibrations

- Pivot calibrations allow a coordinate system to be established with respect to existing data and the world frame.

### 7.2 People

#### Software Development

- Andrew Hundt
- Alex Strickland

### 7.3 Quick Start

#### First Steps

The following steps will show you how to

- download and install CIS on your system.
- use the installation to create an example.
- build and test the example project.

You need to have a Unix-like operating system such as Linux or Mac OS X installed on your machine in order to follow these steps. At the moment, there is no separate tutorial available for Windows users, but you can install CygWin as an alternative. Note, however, that CIS can also be installed and used on Windows.

#### Install CIS

**Get a copy of the source code** Clone the [Git](#) repository from [GitHub](#) as follows:

```
mkdir -p ~/local/src
cd ~/local/src
git clone https://github.com/ahundt/cis
cd cis
```

or [Download](#) a pre-packaged `.tar.gz` of the latest release and unpack it using the following command:

```
mkdir -p ~/local/src
cd ~/local/src
tar xzf /path/to/downloaded/cis-$version.tar.gz
cd cis-$version
```

**Configure the build** Configure the build system using CMake 2.8.4 or a more recent version:

```
mkdir build && cd build
cmake ..
```

- Press `c` to configure the project.
- Change `CMAKE_INSTALL_PREFIX` to `~/local`.
- Set option `BUILD_EXAMPLE` to `ON`.
- Make sure that option `BUILD_PROJECT_TOOL` is enabled.
- Press `g` to generate the Makefiles.

**Build and install CIS** CMake has generated Makefiles for GNU Make. The build is thus triggered by the `make` command:

```
make
```

To install BASIS after the successful build, run the following command:

```
make install
```

As a result, CMake copies the built files into the installation tree as specified by the `CMAKE_INSTALL_PREFIX` variable.

**Set up the environment** For the following tutorial steps, set up your environment as follows. In general, however, only the change of the `PATH` environment variable is recommended. The other environment variables are only needed for the tutorial sessions.

Using the C or TC shell (`csh/tcsh`):

```
setenv PATH "~/local/bin:${PATH}"
setenv CIS_EXAMPLE_DIR "~/local/share/cis/example"
```

Using the Bourne Again SHell (`bash`):

```
export PATH="~/local/bin:${PATH}"
export CIS_EXAMPLE_DIR="~/local/share/basis/example"
```

**Test the Example** Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

```
./cisHWlmain --dataFilenamePrefix pal-debug-a --dataFolderPath /path/to/cis/data/PA1-2/
```

PivotCalibration result **for** `pal-debug-a-empivot.txt`:

```
197.115
192.677
192.437
197.113
192.677
192.434
```



**Command Line Format** The command line format follows standard conventions, plus the ability to store a response file, typically named `*.rsp`, which saves additional command line parameters for future use and convenience. The available command line parameters and descriptions for the primary `cisHW1main` executable file are below.

```
./cisHW1main
```

General Options:

<code>--responseFile arg</code>	File containing additional <code>command</code> line parameters
<code>--help</code>	produce <code>help</code> message
<code>--debug arg (=0)</code>	<code>enable</code> debug output

Algorithm Options:

Data Options:

<code>--dataFolderPath arg (= /Users/athundt/source/cis/build/bin)</code>	folder containing data files, defaults to current working directory
<code>--dataFilenamePrefix arg (=pal-debug-a)</code>	constant prefix of data filename path
<code>--dataFileNameSuffix_calbody arg (--calbody.txt)</code>	suffix of data filename path
<code>--dataFileNameSuffix_calreadings arg (--calreadings.txt)</code>	suffix of data filename path
<code>--dataFileNameSuffix_empivot arg (--empivot.txt)</code>	suffix of data filename path
<code>--dataFileNameSuffix_optpivot arg (--optpivot.txt)</code>	suffix of data filename path
<code>--dataFileNameSuffix_output1 arg (--output1.txt)</code>	suffix of data filename path
<code>--calbodyPath arg</code>	full path to data txt file
<code>--calreadingsPath arg</code>	full path to data txt file
<code>--empivotPath arg</code>	full path to data txt file
<code>--optpivotPath arg</code>	full path to data txt file
<code>--output1Path arg</code>	full path to data txt file

**Unit Test** The easiest way to run the unit test is to build the software, then symlink the data folder “PA1-2” from “data/PA1-2” into the same directory as the unit tests. In other words, the unit tests expect the directory “PA1-2” to be in the same directory as the unit test executable when it is run.

```
ln -s /path/to/cis/data/PA1-2
./cisHW1test
```

## Next Steps

Congratulations! You just finished your first CIS tutorial.

Now check out the [Advanced Information](#) for more details regarding each of the above steps and in-depth information about the used commands if you like, or move on to the various [How-to Guides](#).

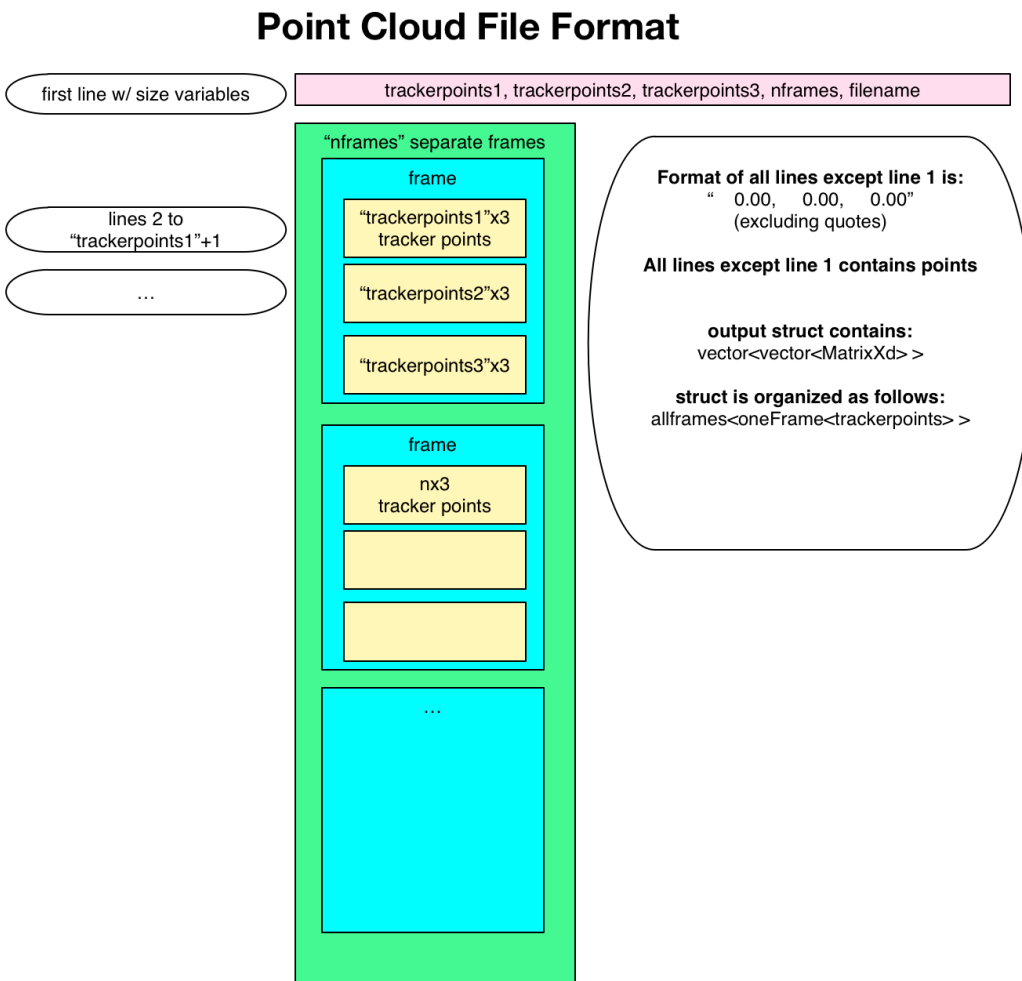
## Advanced Information

For advanced documentation, please see the doxygen API documentation, unit tests, software manual. If you cannot view these files and documents, they are visible as inline source code documentation and and restructured text files found in the `/doc` folder.

For a less comprehensive tutorial-like introduction, please refer to the *First Steps* above.

## 7.4 Read the Data format

The following image illustrates the basics of the input testing data format.



## 7.5 Getting Help

Please report any issues with CIS, including bug reports, feature requests, or support questions, on [GitHub](#).

## 7.6 Reference

### Source Package

<b>config/</b>	Package configuration files.
<b>data/</b>	Data files required by the software.
<b>doc/</b>	Documentation source files.
<b>example/</b>	Example files for users to try out the software.
<b>include/</b>	Header files of the public API of libraries.
<b>lib/</b>	Module files for scripting languages.
<b>modules/</b>	Project modules (i.e., subprojects).
<b>src/</b>	Source code files.
<b>test/</b>	Implementations of unit and regression tests.
AUTHORS.md	A list of the people who contributed to this software.
BasisProject.cmake	Sets basic project information and lists external dependencies.
CMakeLists.txt	Root CMake configuration file.
COPYING.txt	The copyright and license notices.
INSTALL.md	Build and installation instructions.
README.md	Basic summary and references to the documentation.

## 7.7 Download

### Source Code

The source code of the CMake BASIS package is hosted on [GitHub](#) from which all releases and latest development versions can be downloaded. See the [changelog](#) for a summary of changes in each release.

Either clone the Git repository:

```
git clone https://github.com/schuhschuh/cis.git
```

or download a pre-packaged `.tar.gz` of the latest BASIS release:

- [Download CIS v1.0.0 as .tar.gz](#)
- [Download CIS v1.0.0 as .zip](#)

**See also:**

The [Quick Start Guide](#) can help you get up and running.

### System Requirements

**Operating System:** Linux, Mac OS X, Microsoft Windows

### Software License

Copyright (c) 2014 Andrew Hundt and Alex Strickland All rights reserved.

See COPYING file for license information.

### Documentation

[BASIS Manual](#): Online version of this manual

## 7.8 Installation

See the [BASIS guide on software installation](#) for a complete list of build tools and detailed installation instructions.

### Prerequisites

Dependency	Version	Description
<a href="#">BASIS</a>	3.1.0	Utility to automate and standardize creating, documenting, and sharing software.
Boost	1.56.0+	C++ Library collection for general use
CMake	3.0.0	Build Tools.
Eigen	3.2.0	Linear Algebra Library.

### Configure

1. Extract source files:

```
tar -xzf cis-1.0.0-source.tar.gz
```

2. Create build directory:

```
mkdir cis-1.0.0-build
```

3. Change to build directory:

```
cd cis-1.0.0-build
```

4. Run [CMake](#) to configure the build tree:

```
ccmake -DBASIS_DIR:PATH=/path/to/basis ../cis-1.0.0-source
```

- Press `c` to configure the build system and `e` to ignore warnings.
- Set `CMAKE_INSTALL_PREFIX` and other CMake variables and options.
- Continue pressing `c` until the option `g` is available.
- Then press `g` to generate the [GNU Make](#) configuration files.

### Build

After the configuration of the build tree, the software can be build using [GNU Make](#):

```
make
```

### Test

After the build of the software, optionally run the tests using the command:

```
make test
```

In case of failing tests, re-run the tests, but this time by executing [CTest](#) directly with the `-V` option to enable verbose output and redirect the output to a text file:

```
ctest -V >& cis-test.log
```

and attach the file `cis-test.log` to the issue report.

## Install

The final installation copies the built files and additional data and documentation files to the installation directory specified using the `CMAKE_INSTALL_PREFIX` option during the configuration of the build tree:

```
make install
```

After the successful installation, the build directory can be removed again.