Programmer's Manual

Paul Chang

v.1.0

Contents

1 Introduction

This documentation contains the instructions for two sets of code. The *first* provides a description of the gradient nonlinearity code which takes the measured data and generates the spherical harmonic coefficients for each of the gradient fields. The *second* set of code provides a description of the position calibration code used to calculate the optimal positions of NMR field probes using the coefficients of the gradient fields.

All code was written using Matlab™ R2012b, unless otherwise stated.

1.1 Gradient Nonlinearities

This subproject describes the code used to convert the raw probe-measured data to the spherical harmonic coefficients used to characterise the gradient fields. There are two stages in the processing pipeline; firstly, the raw data are converted to position and frequency data; secondly, these data are used to calculate the decomposition coefficients.

1.2 Position Calibration

There are two versions of the code: a Matlab implementation and a C++ implementation.

General Information:

1. Matlab version (*CalcPosition.m*)
   (a) up to 6th order spherical harmonic approximation
   (b) general optimisation algorithm (requires optimisation toolbox for *fminsearch*)
2. C++ version (*CalcPosition.mexw64*)
   (a) up to 4th order spherical harmonic approximation
   (b) Newton's optimisation algorithm

1.3 Related Documentation

This code requires the **spha.h/.mexw64** files. The documentation for these files can be found in .

1.4 Revisions

Revision no.	Date	Author	Changes
1.0	Jan 20, 2016	Paul Chang	First version of the Programmer's Manual

2 Usage

2.1 Gradient Nonlinearities

Calculating the position and frequency data can be done by running the **ProcessGradFields.m** script. The script requires the raw data to be stored in folders Plane0, Plane1, Plane2 and Plane4. The raw data are stored in text files in a human-readable format.

The required output variables are: *posx/posy/posz* and *datax/datay/dataz*.

The *posx, posy, posz* variables are the positions of the probes in a grid format (generated from *meshgrid*). The positions were estimated by finding the probes with almost zero frequency shift and aligning them to the centre fo the coordinate system.

The *datax, datay, dataz* variables are the normalised frequencies of the probes corresponding to the positions. Normalised frequency are essentially position estimates (if the gradient fields were perfectly linear then these frequencies would have the same values as the positions).

These variables are stored in the *nonlin_halfsph.mat* file so that they do not be regenerated and can immediately be used for the following function.

Calculating the gradient field coefficients can be done by running the following function:

coeffs = **GradFieldsToCoeffs** (numSH, posx, posy, posz, data1, data2, data3)

The *numSH* variable is the number of spherical harmonic function to be used for the decomposition for calculating the coefficients.

The *posx, posy, posz* variables are described above.

The *data1, data2, data3* variables are arrays of the same size as the positions with the normalised frequency data (see *datax, datay, dataz* description above). The *data2* and *data3* variables are optional.

2.2 Position Calibration

Both versions can be called using the same function *CalcPosition* and the same inputs/outputs.

opt_pos = **CalcPosition** (freqs, grads)

The *freqs* input variable is a 3xn matrix where n is the number of probes and the rows are the initial x-, y-, z-positions. These positions can be estimated by assuming linear gradient fields.

The *grads* input variable is a 3xn matrix where each row is the coefficients of the spherical harmonic decomposition of the x-, y- and z-gradients respectively.

Gradient field approximations are given using up to 4th order harmonics in *gradcoeffs25.mat* and up to 6th order harmonics in *gradcoeffs49.mat*. These coefficients are calculated from the gradient nonlinearities scripts described in the previous subsection (2.1).

The *opt_pos* output variable is a 3xn matrix where n is the number of probes and the rows are the optimal x-, y-, z-positions.

3 Architecture

The process flow of processing the data is shown in fig. 1.

The file dependencies are shown in fig. 2.

The C++ code can be recompiled using Matlab (refer to `http://uk.mathworks.com/help/matlab/ref/mex.html`).

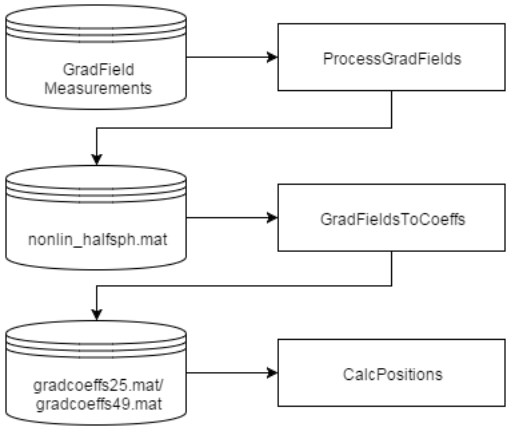

Figure 1: Process flow

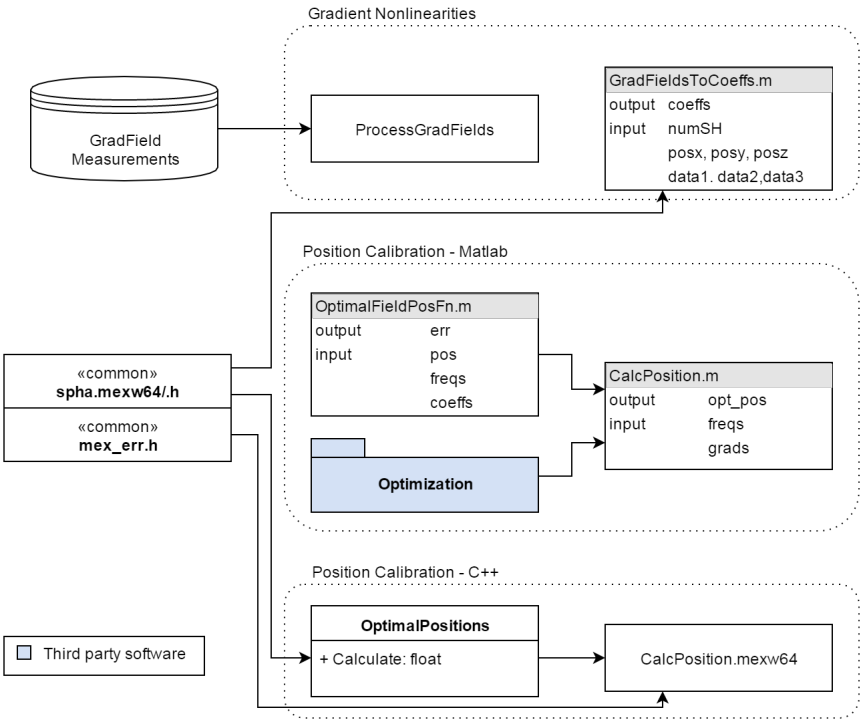

Figure 2: File dependencies

4 Application Program Interface (API) Documentation

These are the functions of the code. The descriptions can be found in the source files.

4.1 Gradient Nonlinearities

- **ProcessGradFields**

  Script used to convert raw data into position and frequency maxtrices.

- coeffs = **GradFieldsToCoeffs** (numSH, posx, posy, posz, data1, [data2, data3])

  Uses the position matrices and decomposes the fields given in *dataX* variables up to *numSH* spherical harmonic functions.

4.2 Position Calibration - Matlab

- opt_pos = **CalcPosition** (freqs, grads)

  Calculates the unconstrained optimal position of probes based on the measured frequencies.

- err = **OptimalFieldPosFn** (pos, freqs, grads)

  Calculates the positions of the probes using measured gradient fields and finding the position that best corresponds to the frequencies of the probes.

4.3 Position Calibration - C++

4.3.1 CalcPosition.cpp

- void **mexFunction** (int *nlhs*, mxArray **plhs*[], int *nrhs*, const mxArray **prhs*[])

  Main function for Matlab mex compilation.

- void **error** (int *err*)

  Error function that interprets the given error code as an error message.

- void **CalcPosition** (double* *opt_pos*, double* *initpos*, int *nprobe*, double* *gradcoeffs*, int *numsh*)

  Calculates the optimal positions for each of the given initial position estimates.

4.3.2 OptimalPosition.h/.cpp

Public Type Definitions

- float *fvec3d*[3]
- float *fmat3d*[3][3]
- vector<float> *fvec*
- float (**basisFn*)(float *x*, float *y*, float *z*)

Public Member Functions

- float **Calculate** (fvec3d *freq*)

  Calculates the optimal position from the initial position estimate (*freq*)

5 Behavioural Description

5.1 Gradient Nonlinearities

5.2 PositionCalib - Matlab

The *CalcPosition* script uses the *OptimalFieldPosFn* function in the *fminsearch* function (from the Optimization toolbox) to calculate the optimal positions.

5.3 PositionCalib - C++

The process flow of the main CalcPosition.cpp file is shown in fig. 3. The functions are also indicated in the diagram. The main process calls on the OptimalPosition class.

The process flow of the Calculate function in the OptimalPosition class are given in fig. 4.

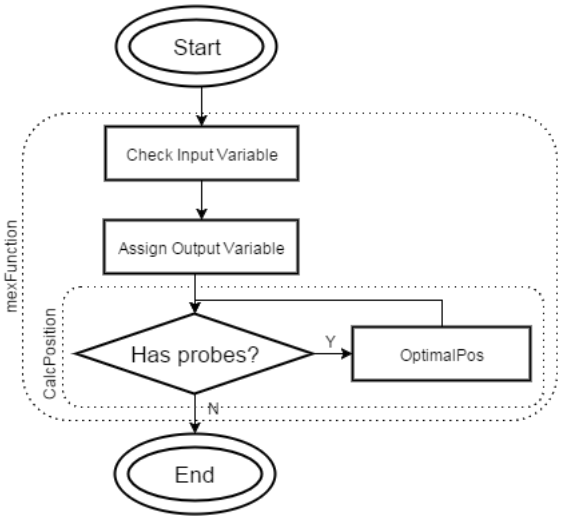

Figure 3: Process flow of CalcPosition.cpp

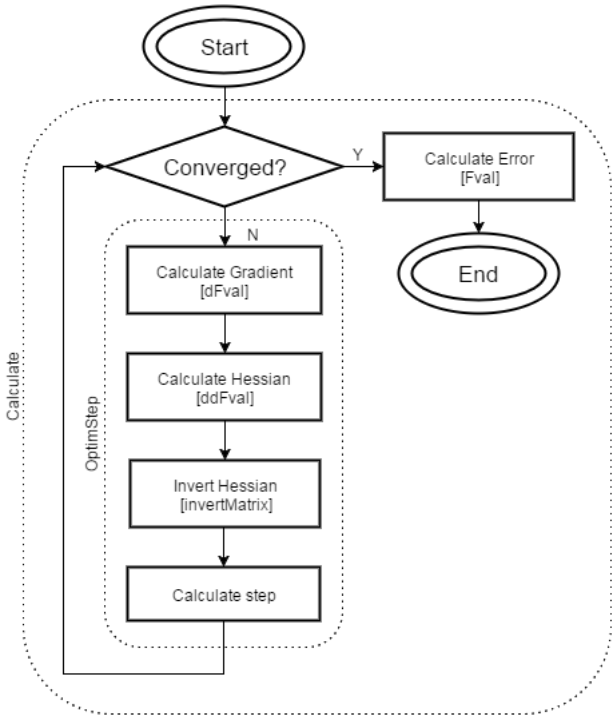

Figure 4: Process flow of Calculate function in OptimalPosition class

6 Output Files

- nonlin_halfsph.mat

  Data variables generated by the ProcessGradFields.m script. The stored variables are posx, posy, posz and datax, datay, dataz.

- gradcoeffs25.mat / gradcoeffs49.mat

  Gradient field coefficients decomposed using up to 4th order and 6th order harmonics respectively. These are generated from the GradFieldsToCoeffs.m script and used by the CalcPosition.m/mexw64 scripts.