simulateDipoleSquareSensorArray

Simulate a sensor array of square shape with dipole magnet as stimulus. Needs options loaded from config file or generated from config generation script. Characterization data must be loaded befor and served as CharData struct. Loops throught positions saves a data set for every supported position of UseOptions which is called TrainingOptions or TestOptions in config.

Contents

Syntax

```
simulateDipoleSquareSensorArray(GeneralOptions, PathVariables, ...
    SensorArrayOptions, DipoleOptions, UseOptions, CharData)
```

Description

simulateDipoleSquareSensorArray(GeneralOptions, PathVariables, ... SensorArrayOptions, DipoleOptions, UseOptions, CharData) saves simulation datasets to data path specified in PathVariables and UseOptions.

Examples

```
% load config from mat-file
load('config.mat', 'GeneralOptions', 'PathVariables', 'SensorArrayOptions',
    'DipoleOptions', 'TrainingOptions', 'TestOptions');

% load characteriazation dataset
TDK = load(PathVariables.tdkDatasetPath);

% generate training dataset(s)
simulateDipoleSquareSensorArray(GeneralOptions, PathVariables, ...
    SensorArrayOptions, DipoleOptions, TrainingOptions, TDK)

% generate test dataset(s)
simulateDipoleSquareSensorArray(GeneralOptions, PathVariables, ...
    SensorArrayOptions, DipoleOptions, TestOptions, TDK)
```

Input Argurments

GeneralOptions struct of general options generate by config script, includes date format and so on.

PathVariables struct of project path generated by config script, includes data path for save and load data.

SensorArrayOptions struct of sensor array shape and behavior generated by congig script.

DipoleOptions struct of dipole specification, defines magnet and stimulus, generated by config script.

UseOptions struct of implemetnation of use case, defines which kind of datset will be generated. At current state test and training dataset are available options in config. In config generated structs are TestOptions and TrainingOptions.

CharData struct of characteriazation data. Therfore load characterization dataset as shown in examples into a struct.

Output Argurments

None

Requirements

- Other m-files required: computeDipoleH0Norm.m, computeDipoleHField.m, genertateDipoleRotationMoments.m, generateSensorArraySquareGrid.m, rotate3DVector.m
- Subfunctions: reshape, interp2, sum
- MAT-files required: config.mat, TDK_TAS2141_Characterization_2020-10-22_18-12-16-827.mat

See Also

- computeDipoleH0Norm
- computeDipoleHField
- genertateDipoleRotationMoments
- generateSensorArraySquareGrid
- rotate3DVector


```
function simulateDipoleSquareSensorArray(GeneralOptions, PathVariables, ...
    SensorArrayOptions, DipoleOptions, UseOptions, CharData)
    arguments
        % validate inputs as struct, structs generated in config.mat
        GeneralOptions struct {mustBeA(GeneralOptions, 'struct')}
        PathVariables struct {mustBeA(PathVariables, 'struct')}
        SensorArrayOptions struct {mustBeA(SensorArrayOptions, 'struct')}
        DipoleOptions struct {mustBeA(DipoleOptions, 'struct')}
        UseOptions struct {mustBeA(UseOptions, 'struct')}
        CharData struct {mustBeA(CharData, 'struct')}
    end

    % try to load relavant values in local variable space for better
    % handling and short names second check if struct fields are reachable
    try
        % general options needed to create filenames etc.
        dfStr = GeneralOptions.dateFormat;

        % number of sensors at edge of square array, dimension N x N
        N = SensorArrayOptions.dimension;
        % sensor array edge length, square edge a
        a = SensorArrayOptions.edge;
        % sensor array supply voltage used to generate bridge outputs from
        % characterization data in combination with bridge offset voltage
        % characterization data should be in mv/V so check norm factor
        Vcc = SensorArrayOptions.Vcc;
        Voff = SensorArrayOptions.Voff;
        Vnorm = SensorArrayOptions.Vnorm;
        switch CharData.Info.Units.SensorOutputVoltage
            case 'mV/V'
                if Vnorm ~= 1e3
                    error('Wrong norming mV/V: %e', Vnorm);
                end
            otherwise
                error('Unknown norm voltage: %s', ...
                    CharData.Info.Units.SensorOutputVoltage)
        end

        % sphere radius for dipole approximation of spherical magnet
        rsp = DipoleOptions.sphereRadius;
        % H-field magnitude to imprint in certain distance from magnet
        % surface which sphere radius rsp plus distance z0
        H0mag = DipoleOptions.H0mag;
        % distance from magnet surface where to imprint the H0mag
        z0 = DipoleOptions.z0;
        % magnetic dipole moment magnitude which define orgin moment of the
        % magnet in rest position
        M0mag = DipoleOptions.M0mag;
```

```
        % dataset type or use case in which later it is use in application
        useCase = UseOptions.useCase;
        % destination path and filename to save gernerated data sets with
        % timestamps in filename, place timestamps with sprintf
        switch useCase
            case 'Training'
                fPath = PathVariables.trainingDataPath;
                fNameFmt = 'Training_%s.mat';
            case 'Test'
                fPath = PathVariables.testDataPath;
                fNameFmt = 'Test_%s.mat';
            otherwise
                error('Unknown use case: %s', UseOptions.useCase);
        end
        % x, y and z positions in which pairing the datasets are generated
        % position vectors are run through in all combinations with tilt
        % and number of angles
        xPos = UseOptions.xPos;
        yPos = UseOptions.yPos;
        zPos = UseOptions.zPos;
        tilt = UseOptions.tilt;
        nAngles = UseOptions.nAngles;
        % constants for generated use case, angle resolution for generated
        % rotation angles, phase index for a phase shift in generation of
        % rotation angles
        angleRes = UseOptions.angleRes;
        phaseIndex = UseOptions.phaseIndex;
        % which characterization reference should be load from CharData
        % sensor output bridge fields (cos/sin)
        refImage = UseOptions.BridgeReference;

        % load values from characterization dataset
        % scales of driven Hx and Hy amplitudes in characteriazation
        % stimulus in kA/m
        if ~strcmp(CharData.Info.Units.MagneticFieldStrength, 'kA/m')
            error('Wrong H-field unit: %s', ...
                CharData.Info.Units.MagneticFieldStrength);
        end
        HxScale = CharData.Data.MagneticField.hx;
        HyScale = CharData.Data.MagneticField.hy;
        % cosinus and sinus characterization images for corresponding field
        % amplitudes, load and norm to Vcc and Voff, references of
        % simulation, adjust reference to bridge gain for output volgates
        gain = CharData.Info.SensorOutput.BridgeGain;
        VcosRef = CharData.Data.SensorOutput.CosinusBridge.(refImage) .* (gain * Vcc / Vn
orm) + Voff;
        VsinRef = CharData.Data.SensorOutput.SinusBridge.(refImage) .* (gain * Vcc / Vnor
m) + Voff;
    catch ME
        rethrow(ME)
    end

    % now everything is successfully loaded, execute further constants
    % which are needs to be generated once for all following operations
    % meshgrids for refernece images to query bridge reference with interp2
    [HxScaleGrid, HyScaleGrid] = meshgrid(HxScale, HyScale);
    % allocate memory for results of on setup run, speed up compute by 10
    % fix allocations which are not changing by varring parameters like
    % number of angles or positon, for all parameter depended memory size
    % allocalte matlab automatically by function call or need reallocation
    % in for loops
    % H-field components for each rotation step
    Hx = zeros(N, N, nAngles);
    Hy = zeros(N, N, nAngles);
    Hz = zeros(N, N, nAngles);
    % H-field abs for each rotation setp
    Habs = zeros(N, N, nAngles);
    % Bridge output voltages for each sensor in grid, H-fields, sensor
    % grid, voltages all same orientation
    Vcos = zeros(N, N, nAngles);
```

```
Vsin = zeros(N, N, nAngles);


% compute values which not changing by loop parameters
% magnetic dipole moments for each rotation step
% rotation angles to compute
% index corresponding to full scale rotation with angleRes
[m, angles, angleRefIndex] = generateDipoleRotationMoments(M0mag, ...
    nAngles, tilt, angleRes, phaseIndex);

% rotation angle step width on full rotation 360° with subset of angles
if length(angles) > 1
    angleStep =  angles(2) - angles(1);
else
    angleStep = 0;
end

% compute dipole rest position norm to imprint a certain field
% strength magnitude with respect of tilt in y axes and magnetization
% in x direction as in generate Dipole rotation moments
r0 = rotate3DVector([0; 0; -(z0 + rsp)], 0, tilt, 0);
m0 = rotate3DVector([-M0mag; 0; 0], 0, tilt, 0);
H0norm = computeDipoleH0Norm(H0mag, m0, r0);

% prepare file header Info struct, overwrite certain fields in loop like x,
% y, z positions
Info = struct;
Info.SensorArrayOptions = SensorArrayOptions;
Info.SensorArrayOptions.SensorCount = N^2;
Info.DipoleOptions = DipoleOptions;
Info.UseOptions = UseOptions;
Info.CharData = join([CharData.Info.SensorManufacturer, CharData.Info.Sensor]);
Info.Units.SensorOutputVoltage = 'V';
Info.Units.MagneticFieldStrength = 'kA/m';
Info.Units.Angles = 'degree';
Info.Units.Length = 'mm';

% collect relevant to Data struct for save to file with file
% header Info struct, overwrite position depended fields in loop before save
Data = struct;
Data.HxScale = HxScale;
Data.HyScale = HyScale;
Data.VcosRef = VcosRef;
Data.VsinRef = VsinRef;
Data.Gain = gain;
Data.r0 = r0;
Data.m0 = m0;
Data.H0norm = H0norm;
Data.m = m;
Data.angles = angles;
Data.angleStep = angleStep;
Data.angleRefIndex = angleRefIndex;

% generate dataset for all use case setup pairs in for loop and append
% generated dataset path to path struct for result
% outer to inner loop is positions to angles
% generate z layer wise
for z = zPos
    for x = xPos
        for y = yPos
            % generate sensor array grid according to current position
            % current position vector of sensor array relative to
            % magnet surface
            p = [x; y; z];
            % write current position in file header
            Info.UseOptions.xPos = x;
            Info.UseOptions.yPos = y;
            Info.UseOptions.zPos = z;
            % sensor array grid coordinates
            [X, Y, Z] = generateSensorArraySquareGrid(N, a, p, rsp);
            % save current sensor gird to Data struct
```

```
                Data.X = X;
                Data.Y = Y;
                Data.Z = Z;
                for i = 1:nAngles
                    % calculate H-field of one rotation step for all
                    % positions, the field is normed to zero position
                    H = computeDipoleHField(X, Y, Z, m(:,i), H0norm);
                    % separate parts or field in axes direction/ components
                    Hx(:,:,i) = reshape(H(1,:), N, N);
                    Hy(:,:,i) = reshape(H(2,:), N, N);
                    Hz(:,:,i) = reshape(H(2,:), N, N);
                    Habs(:,:,i) = reshape(sqrt(sum(H.^2, 1)), N, N);
                    % get bridge outputs from references by cross pick
                    % references from grid, the Hx and Hy queries can be
                    % served as matrix as long they have same size and
                    % orientation the nearest neigbhor interpolation
                    % returns of same size and related to orientation, for
                    % outlayers return NaN, do this for every angle
                    Vcos(:,:,i) = interp2(HxScaleGrid, HyScaleGrid, VcosRef, ...
                        Hx(:,:,i), Hy(:,:,i), 'nearest', NaN);
                    Vsin(:,:,i) = interp2(HxScaleGrid, HyScaleGrid, VsinRef, ...
                        Hx(:,:,i), Hy(:,:,i), 'nearest', NaN);
                end % angles
                % save rotation results to Data struct
                Data.Hx = Hx;
                Data.Hy = Hy;
                Data.Hz = Hz;
                Data.Habs = Habs;
                Data.Vcos = Vcos;
                Data.Vsin = Vsin;
                % save results to file
                fName = sprintf(fNameFmt, datestr(now, dfStr));
                Info.filePath = fullfile(fPath, fName);
                disp(Info.filePath)
                save(Info.filePath, 'Info', 'Data', '-v7.3', '-nocompression');
            end % y
        end % x
    end % z
end
```

Published with MATLAB® R2020b