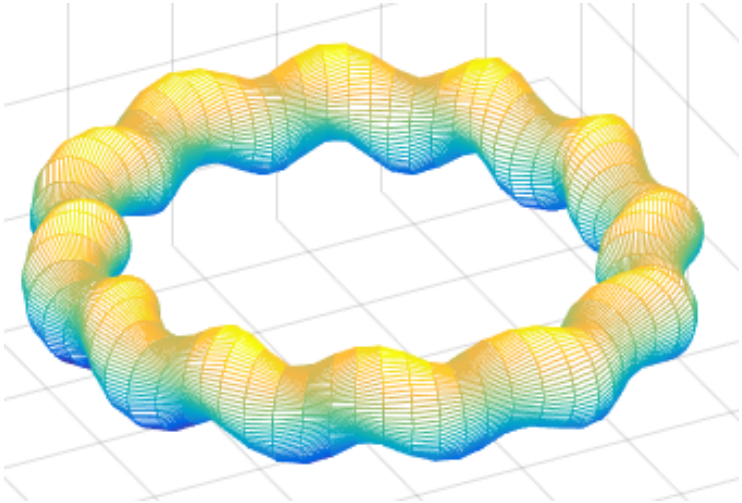
Determination of plasma column shape from the external magnetic field

Henrik Ljunggren

May 31, 2016



Abstract

This report is about creating a program capable of visualizing the plasma column shape using radial flux sensor information and calculating in the frequency domain the Fourier coefficients for the radial flux surface displacement. This program was done in Matlab 2014b. The created GUI gave good results in visualizing the plasma that corresponded with actual testing on the EXTRAP T2R at the fusion plasma physics department at the Royal Institute of Technology.

Contents

1 Introduction

The total world energy need is only becoming larger as more and more people climb out of poverty. In Europe an average person continuously consume 5kW of power which is more than twice that of rest of the world.[1] On top of this is, our living standards are as energy demanding as ever and there is no indication of it slowing down. There is clearly a problem but, what is the solution. The answer to that question is impossible to answer but according to [1] one answer is having a palette of complementing energy sources in which fossil fuels are excluded. The article points out that conventional renewable energy sources will not be enough, and that the final missing piece of the puzzle is indeed harnessing fusion energy. This report is about a tiny step in the journey of world salvation.

1.1 MHD instabilities

In many cases a plasma can be treated as a conductive fluid and its stability analyzed with magneto-hydro-dynamics, MHD. MHD theory is the simplest representation of a plasma. MHD stability is a necessity for a plasma to be stable enough to be used for nuclear fusion, specifically magnetic fusion energy. There exist many different types of instabilities and it's a constant battle to suppress these instabilities or more commonly referred to as perturbations otherwise the plasma quickly dies. Below is but a few of these instabilities,

- Cyclotron instabilities
- Sausage instability
- Tearing mode instability
- Weak beam instability
- Kink instability

An important instability to mention, which is what this report mainly focuses on is the so called kink instability. The main idea on how this works is that where perturbed field lines come together, magnetic pressure is increased and where perturbed field lines come apart, magnetic pressure is decreased. Magnetic pressures then continue to drive the perturbation. No matter which instability, the end effect will be that small perturbations will find ways to increase with nothing suppressing the morphing of the plasma. Looking at figure 1, top right picture, this is an example of an instability, one which will be visible in the upcoming results.

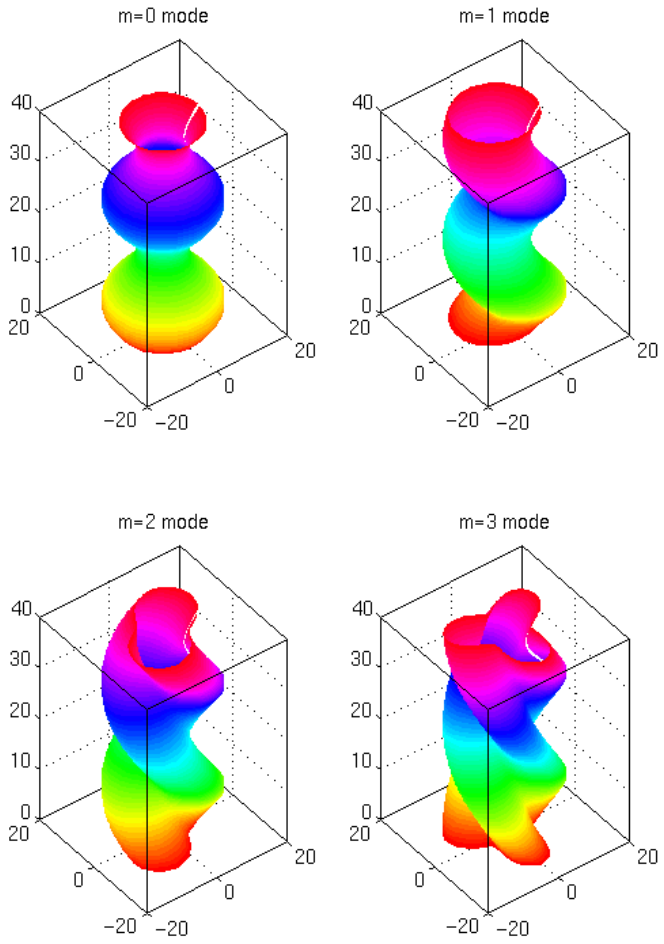


Figure 1: Poloidal mode instabilities

2 Objective

The aim of this report is to show how one can determine the shape of the plasma column from measurements of the external magnetic field. This will be done by gathering the data from the 4x32 radial magnetic flux sensor loops positioned evenly around a tokamak, see figure 2. Using this data the shape of the boundary of the plasma column can be determined.

The reactor used will be the EXTRAP T2R with parameters as defined below:

- Minor radius $r_a = 0.1965$ m
- Major radius $R = 1.24$ m

The plasma column is particularly important in experiments where external control coils are used to keep the plasma instabilities in check. For controlling

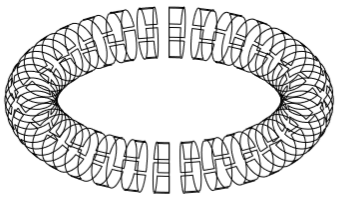


Figure 2: Overveiw of the 4x32 magnetic flux sensors mounted on the EXTRAP T2R

any system, a good measurement of the current state is crucial to obtain good control performance. To tackle the task of visualizing the plasma MATLAB will be used. All data needed will be provided by connecting to a server which has all the sensor information for each conducted experiment.

3 Method

3.1 Sensor information acquirement

Having many sensors can both be a blessing and a curse, in the case of the EXTRAP T2R, in order to reduce required data acquisition channels from 256 to 128, the poloidally opposite magnetic coil outputs are connected in series forming pairs. Sensor information is indeed lost but because the mode $m = 1$ and not the mode $m = 0$ is of interest in the present experiment, we can afford loosing this data. Now 2x32 sensor data will given as the difference of the pair as shown below:

- Coil A output = coil 1 - coil 3
- Coil B output = coil 2 - coil 4

The measured sensor value is the radial magnetic flux which will be converted to radial magnetic field strength by,

$$b_A = \frac{RC}{GA} V_{fluxA} \tag{1}$$

$$b_B = \frac{RC}{GA} V_{fluxB}. \tag{2}$$

Here Rc is the integration time constant of the tokamak ($0.625ms$), G is the preamplifier gains and A the flux loop area ($3.7 \times 10^{-2} m^2$).

In order to fully determine the plasma column shape, both the toroidal and poloidal magnetic fields need to be measured. The poloidal field is created by the plasma current, see figure 3. This magnetic field encapsulates the plasma. The current which drives this magnetic field as well as heats up the plasma is in turn created by coils through induction. In order to calculate the field strength from this plasma current, Ampère's circuital law is used, see below [2]. Measuring the plasma current is not as easy as it seems. To help do this special Rogowski coils are used to measure the current, this makes the measurement independent of the distribution of the current within the loop.

$$\oint_C B_p \, dl = \mu_0 I. \tag{3}$$

Here the closed loop path is the poloidal outline which has the length $2\pi r_a$ and so the equation becomes,

$$B_p = \frac{\mu_0 I}{2\pi r_a}. \tag{4}$$

In the end the current is what is measured and then the resulting magnetic field is calculated.

The second magnetic field of importance, the toroidal field, this field is generated by the many coils fitted around the tokamak, see figure 3. The field strength is measured indirectly by measuring the current flowing through the conductors supplying the toroidal field coils. This is however already computed and stored in the server.

Finally when these two fields combine they form helical shaped field lines which swirl along with the torus, see figure 3. The helical shaped magnetic

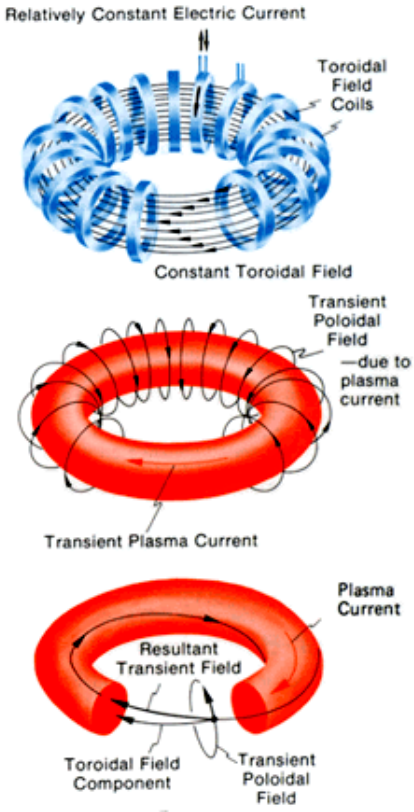


Figure 3: Shown at the top is the toroidal field and the coils needed to create the field. Shown in the middle is the resulting poloidal field from the internal plasma current. Shown at the bottom is the resulting field when both poloidal and toroidal fields are overlaid.

field lines helps keeping the plasma drift in check. The plasma drift or plasma diffusion is yet another problem with keeping the plasma confined and stable. Because of the radial gradient in the field strength of the toroidal field, the positive and negative particles will drift vertically out of position. This will cause the plasma to loose energy to the tokamak wall. In order to counter this, adding the poloidal field is the key which makes the field lines go in a helical path instead of going straight. This causes the particles to drift in the inverse direction back and forth, keeping the particles from hitting the wall. The faster the field lines and in turn the particles rotate the less drifting occurs but the stronger the MHD instabilities become. This is a limit of the tokamak design which won't be discussed here [3].

A big problem with having many different data sources and sensors is how they often don't have the matching sampling frequencies. The measurements of the plasma current need not necessarily coincide with those of the flux sensors loops. To overcome this sensor-fusion problem the data streams need to be shifted such that they line up and appropriate time steps in the simulation should be used. The time step taken during simulation can not be smaller than

the smallest time step of the measurements. In the case of the EXTRAP T2R, the data of the plasma current start much later than the flux loops measurements. Information about the parameters concerning the measurements can be seen in table 1.

Type	**Start time**	**End time**	**Time res**
Flux sensors	-50 ms	962.7 ms	0.1 ms
Plasma current	-12 ms	151.8 ms	0.02 ms
Toroidal field	-12 ms	151.8 ms	0.02 ms

Table 1: Sensor timing information

It is clear that the plasma current and the toroidal field are sampled much faster, but in order to update the visualization at this speed, old data from the flux sensors need be used. Doing this will cause the visualization to change drastically every time 0.1 ms, when five time steps have passed and only slightly every 0.02 ms, on time step. To overcome this problem, taking a longer time step, such as the sampling time for the flux sensors to be exact, will synchronize the visualization. This was done for this project.

3.2 Fast Fourier transform

If the data is assumed to be synchronized then the next step is to do a two dimensional discrete Fourier analysis of the radial magnetic flux. In Matlab this can be done using the fft2() command. A Fourier transform takes target data and converts it into the frequency domain. The input data used is recorded at the same time but not at the same location, this if fed into the fft2() function will then output the magnitude of the spatial frequencies or also called mode numbers. Mode number 1 corresponds to having one period with a phase angle around the torus.

The radial magnetic field strength, b will be used as input for this function. Previously the signals were merged from 4 to only 2 signals. Here 2 dummy signals will be created such that it keeps its spatial structure. This is done as shown below with simData.Brad_A being the first merged signal and simData.Brad_B being the second.

```
b = zeros(4,32);
b(1,:) = simData.brad_A;
b(2,:) = simData.brad_B;
b(3,:) = -simData.brad_A;
b(4,:) = -simData.brad_B;
```

The output of the fft2() is the Fourier mode coefficients. These are given in a 2 dimensional matrix and with rows being, in our case the toroidal modes ordered as n = 0,1,2...14,15,-16,-15...-2,-1 (length 32) and with columns being the poloidal modes ordered as m = -2,-1,0,1 (length 4). The command fftshit() can then be used to reorder the coefficients as n = -16,-15,...0,1,2,...14,15 and m = -2,-1,0,1, see the code snippet below to see how this was done.

```
B = zeros(size(b));
```

Figure 4: Symbolizes the relation between the magnetic field vectors and the flux surface displacement

```
B(:,:) = fftshift(fft2(b(:,:)));
B(:,17) = 0;                % Cancel out the static displacement
ftoroidal = -16:15;         % 0-centered frequency range, n-modes
fpoloidal = [-2,-1,0,1];    % m-modes
```

It is much easier while in the frequency domain to calculate the flux surface displacement from the radial magnetic field strength. By using the similarities of the triangles in figure 4 the following equation can be stated,

$$B_r = \frac{B_\theta}{a}\frac{\delta r_s}{\delta \theta} + \frac{B_\phi}{R}\frac{\delta r_s}{\delta \phi}. \tag{5}$$

To begin with an expression for B_r is needed. This is obtained by an inverse Fourier transformation as shown below,

$$B_r = \sum_m \sum_n C_{mn} e^{i(m\theta + n\phi)} \tag{6}$$

Here the C_{mn} is the Fourier coefficients of the radial magnetic field strength which was calculated before. Next step is expressions for the two partial derivatives of the radial flux displacement surface. Much like before we start with expressing the inverse Fourier transformation of the radial flux displacement surface and at the same time differentiate it with respect to θ and ϕ, which gives,

$$r_s = \sum_m \sum_n S_{mn} e^{i(m\theta + n\phi)} \tag{7}$$

$$\frac{\delta r_s}{\delta \theta} = \sum_m \sum_n S_{mn} ime^{i(m\theta + n\phi)} \tag{8}$$

$$\frac{\delta r_s}{\delta \phi} = \sum_m \sum_n S_{mn} ine^{i(m\theta + n\phi)} \tag{9}$$

Here r_s is the radial flux displacement surface. Combining equations from 5 to 9 results in the final expression,

$$r_s = \sum_m \sum_n \frac{C_{mn}}{i(m\frac{B_\theta}{a} + n\frac{B_\phi}{R})} e^{i(m\theta + n\phi)} \tag{10}$$

Using this equation makes it possible to interpolate in between points, a finer resolution than the original 4x32 can be achieved. The Matlab script runs

through all the specified angles and saves the corresponding r_s. It is important to remember that when creating the angle resolution vector to exclude the last point so that no duplicates exist. This is how it was done in this project,

```
thetaRes =
    -pi:(2*pi/round(simData.theta_res)):(pi-(2*pi/round(simData.theta_res)));
phiRes =
    -pi:(2*pi/round(simData.phi_res)):(pi-(2*pi/round(simData.phi_res)));
```

Here simData.theta_res is how many poloidal segments will be used and simData.phi_res how many toroidal segments.

3.3 Plotting

Plotting in Matlab means connection two adjacent data point together with a surface or line if in 3D. Hence we need to close the loop by copying the start to an additional point at the end. The angle resolution now becomes,

```
thetaRes = -pi:(2*pi/round(simData.theta_res)):pi;
phiRes = -pi:(2*pi/round(simData.phi_res)):pi;
```

The data point re-connection from the end to the start again was done as follows,

```
rSize = size(r)+[1 1];
rClosed = zeros(rSize);
rClosed(1:(end-1),1:(end-1)) = r;
rClosed(1:(end-1),end) = r(:,1);
rClosed(end,(1:(end-1))) = r(1,:);
```

At this point all information needed for plotting is available but in order for Matlab to correctly plot the plasma column shape it needs the data to be in the Cartesian coordinate system. The conversion is done as shown below,

```
x(row,col) = (simData.Rmajor + (simData.aminor + rClosed(row,col)/1000)
    *cos(theta)) * cos(phi);
y(row,col) = (simData.Rmajor + (simData.aminor + rClosed(row,col)/1000)
    * cos(theta)) * sin(phi);
z(row,col) = (simData.aminor + rClosed(row,col)/1000)* sin(theta);
```

Finally by simply invoking any 3D plotting such as surf() or mesh() the plasma column can be visualized and by adding the radial flux displacement surface vector a color map will tell you the magnitude of the displacement.

```
surf(x, y, z, r(:,:));
axis([-1.8 1.8 -1.8 1.8 -.8 .8])
```

3.4 GUI

A GUI was created to help with plasma research which has the following capabilities,

- Adjustable resolution for Time, Poloidal angle, Toroidal angle
- Radial flux displacement surface boosting
- Free rotation while simulating
- Stop and play
- Specifying the time frame
- Two plots with time indicator
- Time manipulator
- Capture current image
- Display styles Surf, Mesh, Rendered, Modes, cut view

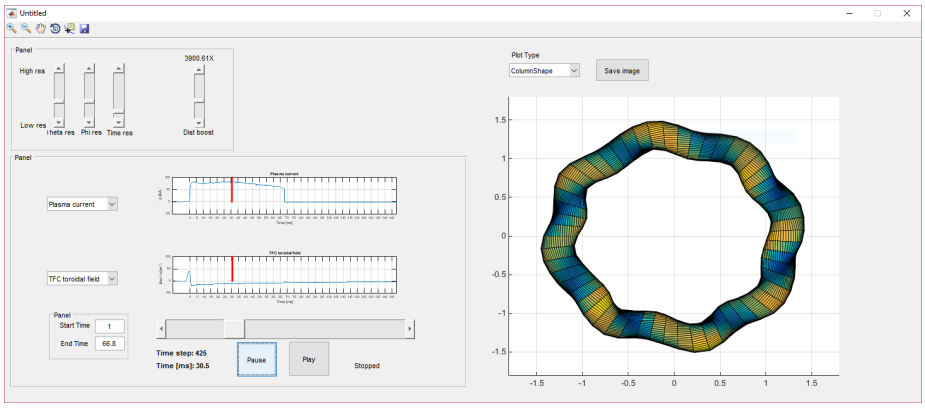


Figure 5: GUI for visualizing the plasma column shape created for this project.

4 Results and Conclusions

Testing of the software by specifying certain characteristics for a plasma run using the MHD control system installed in the EXTRAP T2R. All test results can be found in table 2 and 3 together with a reference to corresponding sample image for each test case. For all images the setting were:

- Image taken at 20 ms
- Radial displacement amplification boost set to 7000
- Angle resolution set to $\frac{2\pi}{55}$ for both poloidal and toroidal angle

4.1 Results

Data ID	Description	Expected results	Actual Results
25734, test 1	Initial test run of the reactor	A plasma shape with no particular structure and average life span	OK: Figure: 6, A slightly fluctuation in the plasma. The plasma died at 74 ms.
25735, test 2	Amplifying the mode number n = 8, amp = 0.01 from 10 ms to 30 ms	A plasma torus with 8 bulges corresponding to n = 8	OK: Figure: 7, a torus shape consisting of 8 periods clearly visible. The plasma died at 75 ms.
25736, test 3	Amplifying the mode number n = 8, amp = 0.02 from 10 ms to 50 ms	A plasma torus with 8 bulges relatively strong corresponding to n = 8	OK: Figure: 8, a torus shape consisting of 8 periods clearly visible, stronger than before. The plasma died at 63 ms.
25737, test 4	Amplifying the mode number n = 8, amp = 0.02 from 10 ms to 50 ms and rotating the plasma with $\omega = 20$	A rotating plasma torus with 8 bulges relatively strong corresponding to n = 8	FAIL: Figure: 9, a torus shape consisting of 8 periods clearly visible, pulsating slowly but no rotation. The plasma died at 75 ms.

Table 2: Testing results 1

Data ID	Description	Expected results	Actual Results
25738, test 5	Amplifying the mode number n = 8, amp = 0.02 from 10 ms to 50 ms and rotating the plasma with $mode_{rot} = 20$.	A rotating plasma torus with 8 bulges relatively strong corresponding to n = 8	OK: Figure: 10, a torus shape consisting of 8 periods clearly visible, slowly rotating counter clockwise when looking from above. The plasma died at 72 ms.
25739, test 6	Amplifying the mode number n = -8, amp = 0.02 from 10 ms to 50 ms and rotating the plasma with $mode_{rot} = 20$.	A rotating plasma torus with 8 bulges relatively strong corresponding to n = 8. It should rotate opposite of test 5	OK: Figure: 11, a torus shape consisting of 8 periods clearly visible, slowly rotating clockwise when looking from above. The helical shape is also twisted in the opposite direction than that of test 5. The plasma died at 45 ms.
25740, test 7	Amplifying the mode number n = 1, amp = 0.02 from 10 ms to 50 ms and rotating the plasma with $mode_{rot} = 20$.	The entire plasma torus should move in a circle counter clockwise	OK: Figure: 12, a circular torus moving very slowly around counter clockwise in a circle. The torus seems slightly distorted and is not completely round in the poloidal cut view. The plasma died at 60 ms.
25742, test 8	Amplifying the mode number n = 12, amp = 0.02 from 10 ms to 50 ms and rotating the plasma with $mode_{rot} = 20$.	A rotating plasma torus with 12 bulges relatively strong corresponding to n = 12. It should rotate counter clockwise.	OK: Figure: 13, a torus shape consisting of 12 periods clearly visible, slowly rotating counter clockwise when looking from above. The plasma died at 50 ms.
25744, test 9	Amplifying the mode number n = 12, amp = 0.02 from 10 ms to 50 ms and rotating the plasma with $mode_{rot} = 100$.	A rotating plasma torus with 12 bulges relatively strong corresponding to n = 12. It should rotate counter clockwise fairly quickly.	OK: Figure: 14, a torus shape consisting of 12 periods clearly visible, quickly rotating counter clockwise when looking from above. The plasma died very quickly at 25 ms.

Table 3: Testing results 2

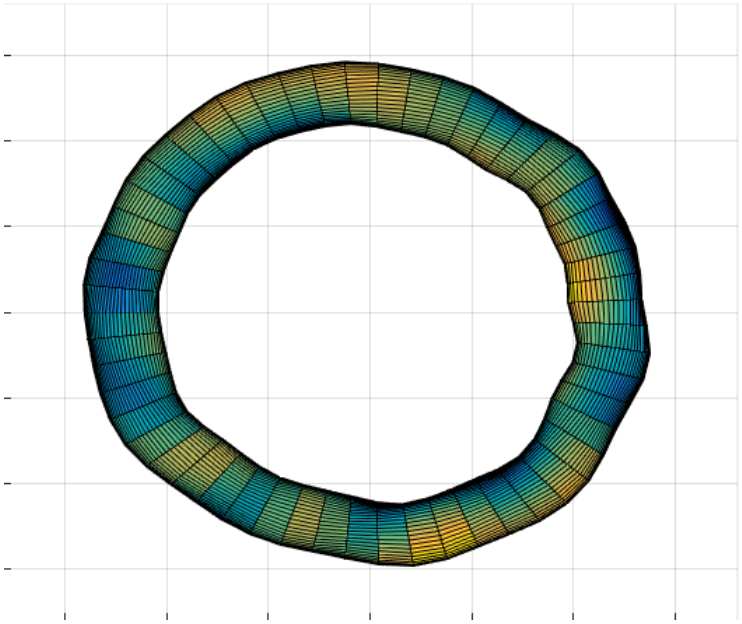


Figure 6: Test case 1, normal operation.

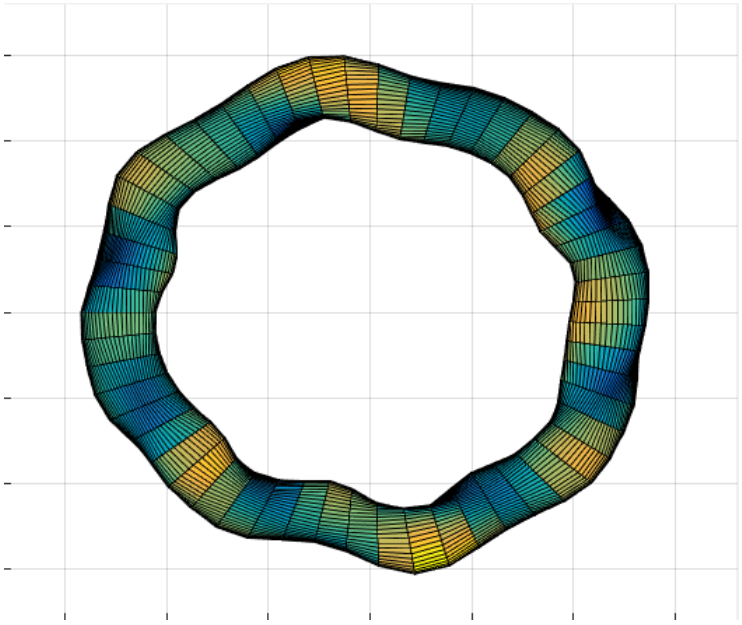


Figure 7: Test case 2, mode amplification = 0.01, n = 8.

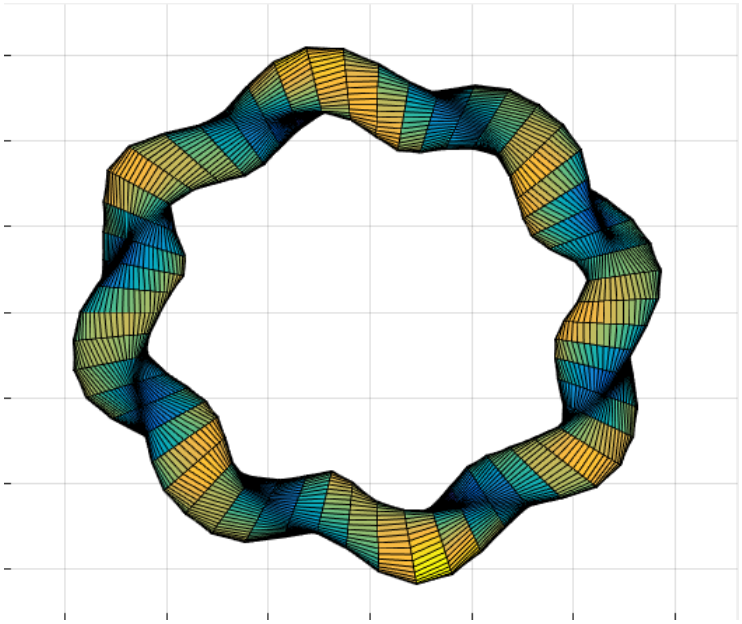


Figure 8: Test case 3, mode amplification = 0.02, n = 8.

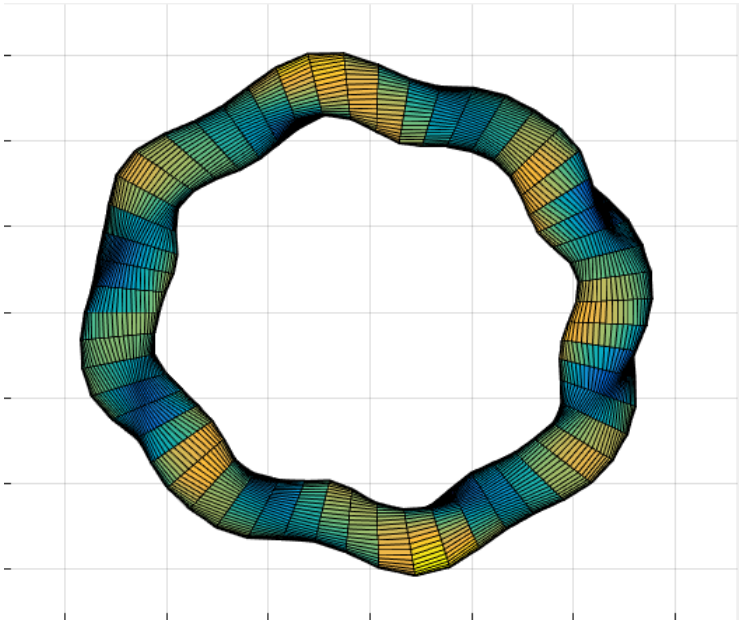


Figure 9: Test case 4, mode amplification = 0.02, n = 8, $\omega = 20$.

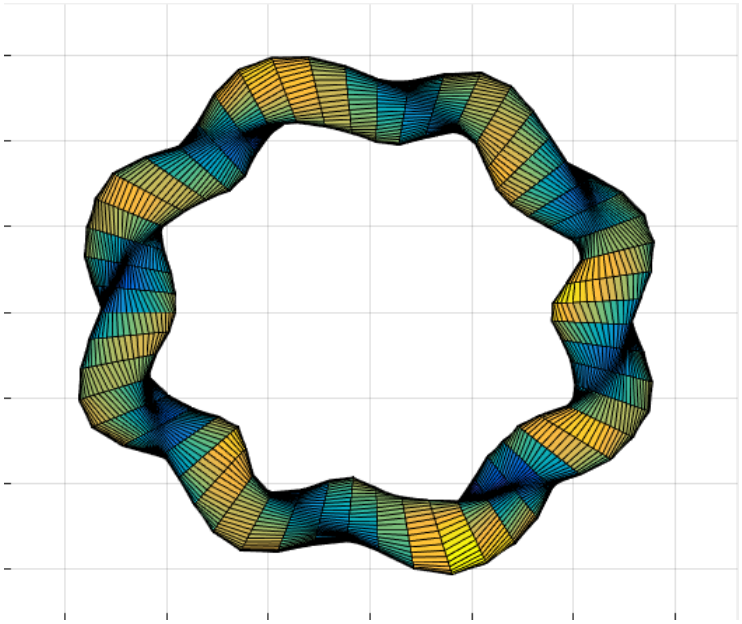


Figure 10: Test case 5, mode amplification = 0.02, n = 8, $mode_{rot} = 20$.

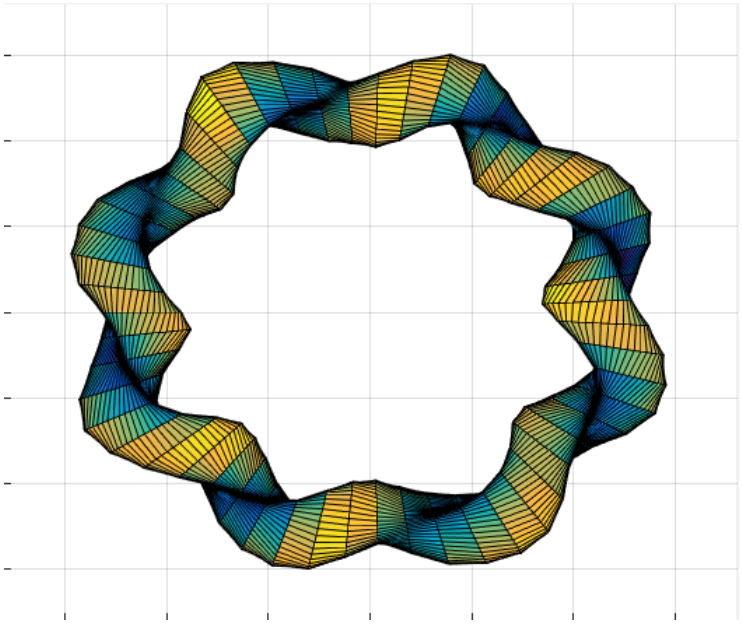


Figure 11: Test case 6, mode amplification = 0.02, n = -8, $mode_{rot} = 20$.

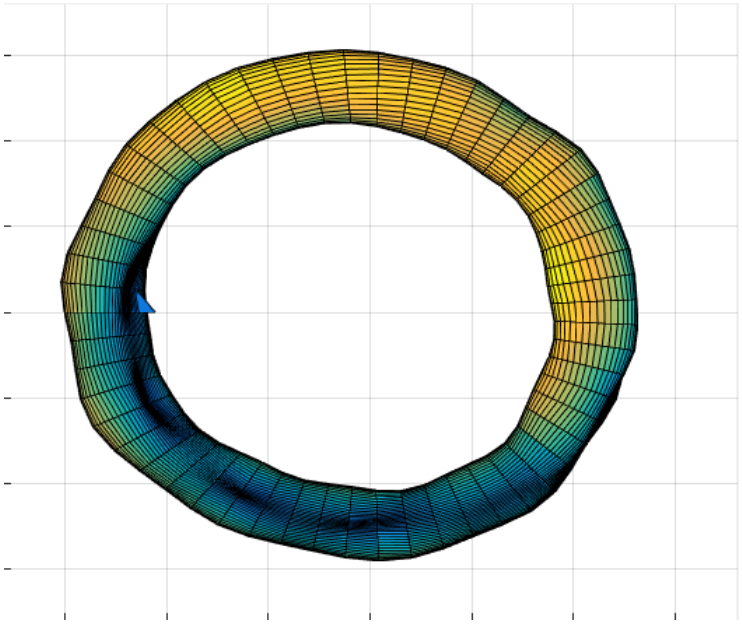


Figure 12: Test case 7, mode amplification = 0.02, n = 1, $mode_{rot} = 20$.

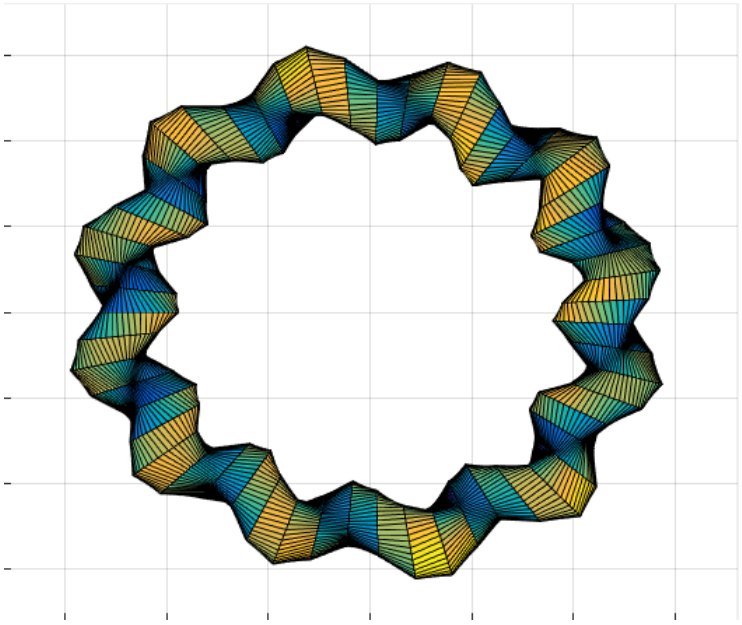


Figure 13: Test case 8, mode amplification = 0.02, n = 12, $mode_{rot} = 20$.

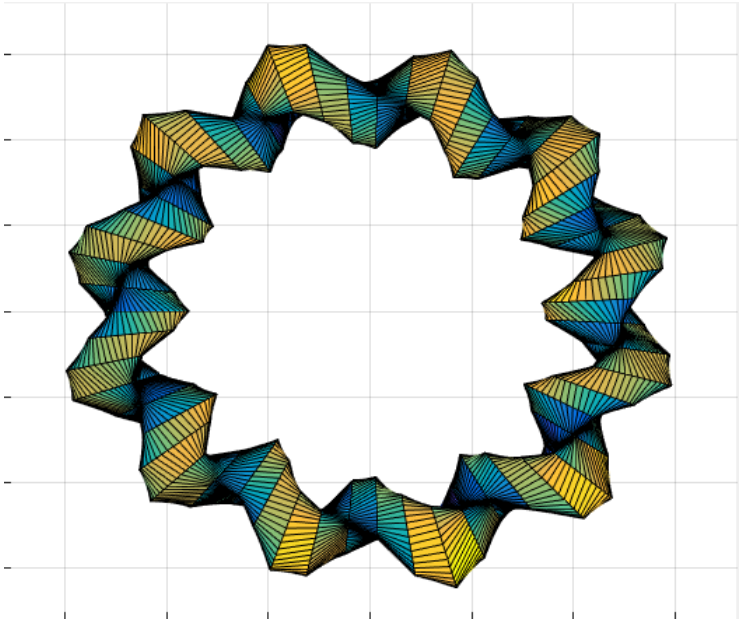


Figure 14: Test case 9, mode amplification = 0.02, n = 12, $mode_{rot} = 100$.

4.2 Conclusions

The expected results seems to coincide with the actual results meaning that the visualization program presents what the MHD control program tries to achieve.

The plasma life span seems longer when having slow rotation and/or low mode number. In test 5 the plasma life span is 72 ms and in test 8 with the same configuration but with higher mode number, n = 12 the life span is 50 ms. In test 8 the rotation speed is $20 rad/s$ but the life span i halved, 25 ms when increasing the speed to $100 rad/s$.

In test 7 it can be seen that the poloidal cross section is not an even circle. This is somewhat of a problem, mainly because previously, poloidal mode numbers larger than m = 1 were discarded. This was done by merging the 4 flux sensor data inputs to only 2 and the later artificially creating 4 inputs again causing modes -2 and 0 to be 0 in magnitude. This kind of deformation should therefor not show in the simulation. This could indicate an error in this program.

5 Discussion and Experimental Uncertainty

Overall the program works perfectly but as mentioned in the conclusion there is a problem with the poloidal deformation. It was by debugging concluded that the output of the fft2(), the magnetic field strength did not fulfill that the conjugate of a mode number should be the same as the negative mode number. This could be a root of the problem given that the program assumes this and when calculating the radial flux displacement only takes the real part in to account. Further work on this will be done in order to fix this problem.

Something not mentioned in this report yet is what happens when no plasma has formed and after it dies. What happens is that in the visualization program the plasma column shape explodes into what could resemble modern art. It does this because either the plasma current or the toroidal field is zero. There can generally not be a plasma current without a plasma. When this happens, part of the calculation, see equation 10 divides by zero or close to zero amplifying the numerical and sensor error to be enormous which in turn causes the fluctuating super nova show in the GUI. With this said, the program is only of use during normal use with no input part being zero or very close to zero.

In order to use this program for research or evaluation of the plasma it needs to be correct. The model to calculate the plasma column here has its limitations and is only an approximation of the reality. For example, not all the modes of the plasma is observable due to the spatial resolution of the sensors and is therefore not included in the visualization. They are thought to be very small and are treated as neglectable. This is true for the toroidal modes but the measurement of the poloidal modes has much lower resolution. These modes most definitely changes the shape of the cross section but are not as crucial for plasma stability as the toroidal ones.

5.1 Future work

Further developments of this program is fixing the poloidal distortion to correspond to the actual input mode numbers. It could maybe also be nice to be able to overlay another set of testing data on the plasma torus such that you could see the difference between test cases.

Another idea is to overlay an approximation of the the future state based on old sensor data. By doing this one could see how good ones simulations algorithm actually is. Taking this a step further, something I personally would like to do is and see if it could work is to create my own control algorithm. This algorithm would be a neural network taught with back-propagation in a MHD test simulation environment. Maybe controlling MHD plasma instabilities is so complicated and the patterns so hard to see that creating an AI utilizing the abstract input patterns could actually work.

References

[1] Jan Scheffel, *Why we need fusion energy*, [Online] `http://people.kth.se/~jans/papers/Why_we_need_fusion.pdf`.

[2] Hyper physics, [Online], `http://hyperphysics.phy-astr.gsu.edu/hbase/magnetic/amplaw.html`.

[3] Per Brunsell, *Experimental fusion plasma physics*, 2014.

Appendices

GUI code

```
function varargout = FusionUI3(varargin)
% FUSIONUI3 MATLAB code for FusionUI3.fig
%      FUSIONUI3, by itself, creates a new FUSIONUI3 or raises the
    existing
%      singleton*.
%
%      H = FUSIONUI3 returns the handle to a new FUSIONUI3 or the handle
    to
%      the existing singleton*.
%
%      FUSIONUI3('CALLBACK',hObject,eventData,handles,...) calls the
    local
%      function named CALLBACK in FUSIONUI3.M with the given input
    arguments.
%
%      FUSIONUI3('Property','Value',...) creates a new FUSIONUI3 or
    raises the
%      existing singleton*. Starting from the left, property value pairs
    are
%      applied to the GUI before FusionUI3_OpeningFcn gets called. An
%      unrecognized property name or invalid value makes property
    application
%      stop. All inputs are passed to FusionUI3_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only
    one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help FusionUI3

% Last Modified by GUIDE v2.5 29-May-2016 11:46:25

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',    mfilename, ...
                   'gui_Singleton', gui_Singleton, ...
                   'gui_OpeningFcn', @FusionUI3_OpeningFcn, ...
                   'gui_OutputFcn', @FusionUI3_OutputFcn, ...
                   'gui_LayoutFcn', [] , ...
                   'gui_Callback', []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
```

```
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before FusionUI3 is made visible.
function FusionUI3_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject   handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)
% varargin  command line arguments to FusionUI3 (see VARARGIN)


%% Setting up handles
handles.currentTimeStep = 1;
handles.contRunningSim = 0;


%% Get Data
mode = 0;
animateModes = 0;

if mode == 0
    %Get local data
%    load('plot_shot_new','-mat')
%    load('plot_br_signals','-mat')

   load('Test8_25742','-mat')

   handles.plas_curr = plas_curr;
   handles.plas_curr_tm = plas_curr_tm;
   handles.loop_volt_tor = loop_volt_tor;
   handles.loop_volt_tor_tm = loop_volt_tor_tm;
   handles.btor_ave = btor_ave;
   handles.btor_ave_tm = btor_ave_tm;
   handles.btor_tfc = btor_tfc;
   handles.btor_tfc_tm = btor_tfc_tm;

   handles.N_coil = N_coil;
   handles.N_time = N_time;
   handles.theta = theta;
   handles.phi = phi;
   handles.time = time;
   handles.brad_A = brad_A;
   handles.brad_B = brad_B;



else
    %Get data from network
    prompt = {'Shot number:'};
    titl = 'Run parameters';
    lines = 1;
    answer = inputdlg(prompt, titl, lines);
```

```
    shot_number = str2double(answer{1});


    [handles.N_coil, handles.N_time, handles.theta, handles.phi,
        handles.time, handles.brad_A, handles.brad_B] = ...
           read_br_signals( shot_number);

   % mdsip server at KTH-5717
    mdsconnect( '130.237.45.47');

    mdsopen( 'T2R', shot_number);

    handles.plas_curr = mdsvalue( '\gbl_itor_pla');
    handles.plas_curr_tm = mdsvalue( 'dim_of( \gbl_itor_pla)');

    handles.loop_volt_tor = mdsvalue( '\gbl_vtor_lin');
    handles.loop_volt_tor_tm = mdsvalue( 'dim_of( \gbl_vtor_lin)');

    handles.btor_ave = 1e3 * mdsvalue( '\gbl_btor_ave');
    handles.btor_ave_tm = mdsvalue( 'dim_of( \gbl_btor_ave)');

    handles.btor_tfc = 1e3 * mdsvalue( '\gbl_btor_tfc');
    handles.btor_tfc_tm = mdsvalue( 'dim_of( \gbl_btor_tfc)');

    mdsclose;

end

% Choose default command line output for FusionUI3
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% This sets up the initial plot - only do when we are invisible
% so window can get raised using FusionUI3.
if strcmp(get(hObject,'Visible'),'off')
    plot(rand(5));
end

% UIWAIT makes FusionUI3 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = FusionUI3_OutputFcn(hObject, eventdata, handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject   handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

```
% --------------------------------------------------------------------
function FileMenu_Callback(hObject, eventdata, handles)
% hObject   handle to FileMenu (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function OpenMenuItem_Callback(hObject, eventdata, handles)
% hObject   handle to OpenMenuItem (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)
file = uigetfile('*.fig');
if ~isequal(file, 0)
    open(file);
end

% --------------------------------------------------------------------
function PrintMenuItem_Callback(hObject, eventdata, handles)
% hObject   handle to PrintMenuItem (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)
printdlg(handles.figure1)

% --------------------------------------------------------------------
function CloseMenuItem_Callback(hObject, eventdata, handles)
% hObject   handle to CloseMenuItem (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)
selection = questdlg(['Close ' get(handles.figure1,'Name') '?'],...
                     ['Close ' get(handles.figure1,'Name') '...'],...
                     'Yes','No','Yes');
if strcmp(selection,'No')
    return;
end

delete(handles.figure1)


% --- Executes on slider movement.
function time_slider_Callback(hObject, eventdata, handles)
% hObject   handle to time_slider (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
    slider

handles.currentTimeStep = round(get(hObject,'Value'));
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
```

```
function time_slider_CreateFcn(hObject, eventdata, handles)
% hObject   handle to time_slider (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   empty - handles not created until after all CreateFcns called
handles.maxCurrentTimeStep = round(8000/5);
set(hObject,'Min',1)
set(hObject,'Max',handles.maxCurrentTimeStep)
set(hObject,'Value',1)
guidata(hObject, handles);

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on selection change in popupmenu2.
function popupmenu2_Callback(hObject, eventdata, handles)
% hObject   handle to popupmenu2 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu2
    contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
    popupmenu2


% --- Executes during object creation, after setting all properties.
function popupmenu2_CreateFcn(hObject, eventdata, handles)
% hObject   handle to popupmenu2 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in play_button.
function play_button_Callback(hObject, eventdata, handles)
% hObject   handle to play_button (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)
% handles.contRunningSim = 1;
set(handles.play_button,'UserData',1)
set(handles.runningStatus,'String','Running')


while get(handles.play_button,'UserData')==1
```

```
    simDataToBeProcessed = pickOutData(handles);


    if simDataToBeProcessed.currentTimeStep >= handles.maxCurrentTimeStep
        set(handles.play_button,'UserData',0);
        handles.currentTimeStep = handles.maxCurrentTimeStep;
        set(handles.runningStatus,'String','Stopped')
    else
        handles.currentTimeStep = simDataToBeProcessed.currentTimeStep;
    end
    guidata(hObject, handles);

    axes(handles.main_axis)
    contents = cellstr(get(handles.main_axis_popup,'String'));
    RunSimulationIncrement(simDataToBeProcessed,contents{get(handles.main_axis_popup,'Value')})

    axes(handles.lower_axis)
    contents = cellstr(get(handles.lower_popup,'String'));
    plotHelpPlot(contents{get(handles.lower_popup,'Value')},simDataToBeProcessed,handles)

    axes(handles.upper_axis)
    contents = cellstr(get(handles.upper_popup,'String'));
    plotHelpPlot(contents{get(handles.upper_popup,'Value')},simDataToBeProcessed,handles)

    set(handles.timeText,'String',['Time step: '
        num2str(handles.currentTimeStep)])
    set(handles.timeText2,'String',['Time [ms]: '
        num2str((handles.currentTimeStep)/10-12)])
    set(handles.amp_boost_value,'String',[num2str(get(handles.amp_boost,'Value'))
        'X'])

    set(handles.time_slider,'Value',handles.currentTimeStep)
    pause(0.05)
end



% --- Executes on button press in pause_button.
function pause_button_Callback(hObject, eventdata, handles)
% hObject   handle to pause_button (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)
set(handles.play_button,'UserData',0)
set(handles.runningStatus,'String','Stopped')
% guidata(hObject, handles);

% --- Executes on selection change in upper_popup.
function upper_popup_Callback(hObject, eventdata, handles)
% hObject   handle to upper_popup (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)
```

```
% Hints: contents = cellstr(get(hObject,'String')) returns upper_popup
    contents as cell array
%       contents{get(hObject,'Value')} returns selected item from
    upper_popup


% --- Executes during object creation, after setting all properties.
function upper_popup_CreateFcn(hObject, eventdata, handles)
% hObject   handle to upper_popup (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in lower_popup.
function lower_popup_Callback(hObject, eventdata, handles)
% hObject   handle to lower_popup (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns lower_popup
    contents as cell array
%       contents{get(hObject,'Value')} returns selected item from
    lower_popup

% --- Executes during object creation, after setting all properties.
function lower_popup_CreateFcn(hObject, eventdata, handles)
% hObject   handle to lower_popup (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in main_axis_popup.
function main_axis_popup_Callback(hObject, eventdata, handles)
% hObject   handle to main_axis_popup (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)
% contents = cellstr(get(main_axis_popup,'String'));
%
% if contents{get(main_axis_popup,'Value')} == 'Freq modes'
```

```
%     axes(handles.main_axis)
%     view(0,0)
% elseif contents{get(main_axis_popup,'Value')} == 'Column shape'
%     axes(handles.main_axis)
%     view(0,0)
% end

% Hints: contents = cellstr(get(hObject,'String')) returns
    main_axis_popup contents as cell array
%       contents{get(hObject,'Value')} returns selected item from
    main_axis_popup


% --- Executes during object creation, after setting all properties.
function main_axis_popup_CreateFcn(hObject, eventdata, handles)
% hObject   handle to main_axis_popup (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on slider movement.
function time_res_Callback(hObject, eventdata, handles)
% hObject   handle to time_res (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
    slider


% --- Executes during object creation, after setting all properties.
function time_res_CreateFcn(hObject, eventdata, handles)
% hObject   handle to time_res (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   empty - handles not created until after all CreateFcns called
set(hObject,'Min',1)
set(hObject,'Max',100)
set(hObject,'Value',10)

guidata(hObject, handles);
% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
```

```
% --- Executes on slider movement.
function theta_res_Callback(hObject, eventdata, handles)
% hObject    handle to theta_res (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
    slider



% --- Executes during object creation, after setting all properties.
function theta_res_CreateFcn(hObject, eventdata, handles)
% hObject    handle to theta_res (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
set(hObject,'Min',1)
set(hObject,'Max',150)
set(hObject,'Value',55)

guidata(hObject, handles);
% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function phi_res_Callback(hObject, eventdata, handles)
% hObject    handle to phi_res (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
    slider


% --- Executes during object creation, after setting all properties.
function phi_res_CreateFcn(hObject, eventdata, handles)
% hObject    handle to phi_res (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
set(hObject,'Min',1)
set(hObject,'Max',150)
set(hObject,'Value',55)

guidata(hObject, handles);
% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes during object creation, after setting all properties.
function theta_res_text_CreateFcn(hObject, eventdata, handles)
% hObject    handle to theta_res_text (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes during object creation, after setting all properties.
function phi_res_text_CreateFcn(hObject, eventdata, handles)
% hObject    handle to phi_res_text (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes on slider movement.
function amp_boost_Callback(hObject, eventdata, handles)
% hObject    handle to amp_boost (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
    slider


% --- Executes during object creation, after setting all properties.
function amp_boost_CreateFcn(hObject, eventdata, handles)
% hObject    handle to amp_boost (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
set(hObject,'Min',1)
set(hObject,'Max',10000)
set(hObject,'Value',7000)

guidata(hObject, handles);
% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



function startTime_Callback(hObject, eventdata, handles)
% hObject    handle to startTime (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of startTime as text
%        str2double(get(hObject,'String')) returns contents of startTime
```

```
    as a double


% --- Executes during object creation, after setting all properties.
function startTime_CreateFcn(hObject, eventdata, handles)
% hObject    handle to startTime (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
set(hObject,'String',num2str(1))
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function endTime_Callback(hObject, eventdata, handles)
% hObject    handle to endTime (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if
    str2double(get(hObject,'String'))<=round(handles.maxCurrentTimeStep*5*0.02-12)
    && str2double(get(hObject,'String'))>=-12
handles.maxCurrentTimeStep =
    (str2double(get(hObject,'String'))+12)/5/0.02+1;
set(handles.time_slider,'Max',handles.maxCurrentTimeStep)

    if handles.currentTimeStep >= handles.maxCurrentTimeStep
        set(handles.time_slider,'Value',handles.maxCurrentTimeStep)
    end

else
set(hObject,'String',num2str(handles.maxCurrentTimeStep*5*0.02-12))
end
guidata(hObject, handles);

set(hObject,'String',num2str(handles.maxCurrentTimeStep))
% Hints: get(hObject,'String') returns contents of endTime as text
%        str2double(get(hObject,'String')) returns contents of endTime
    as a double


% --- Executes during object creation, after setting all properties.
function endTime_CreateFcn(hObject, eventdata, handles)
% hObject    handle to endTime (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
handles.maxCurrentTimeStep = round(8000*5*0.02-12);

guidata(hObject, handles);

set(hObject,'String',num2str(handles.maxCurrentTimeStep*5*0.02-12))
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in saveImage.
function saveImage_Callback(hObject, eventdata, handles)
% hObject   handle to saveImage (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)
F = getframe(handles.main_axis);
Image = frame2im(F);
%TODO: Add parameter name when generating image
imwrite(Image, 'savedImage.png')


guidata(hObject, handles);
```

Simulation code

```
function [ ] = RunSimulationIncrement( simData ,plotTypeString)
%RUNSIMULATION Summary of this function goes here
%   Detailed explanation goes here

%% Equilibrium magnetic field components
mu0 = 4*pi*10e-7;                                   %[N/A^2]
plas_curr_A = simData.plas_curr*1000;               %[A]
Bpoloidal = mu0*plas_curr_A/(2*pi*simData.aminor); %From Ampare's law,
    ?I=circleIntegral(B_vec dot product dl_vec)
% Bpoloidal = mu0*plas_curr_A/(2*pi*simData.Rmajor); %From Ampare's law,
    ?I=circleIntegral(B_vec dot product dl_vec)
Bpoloidal = Bpoloidal*1000;                         %Convert to mT

% Bpoliodal = 9.152293387657957e+02;

%% Creating dummy sensor dimensions
b = zeros(4,32);
b(1,:) = simData.brad_A;
b(2,:) = simData.brad_B;
b(3,:) = -simData.brad_A;
b(4,:) = -simData.brad_B;

%% Fast furier transform
% B = zeros(size(b));
B = fftshift(fft2(b(:,:)));
B(:,17) = 0; %Cancel out the static displacement
ftoroidal = -16:15; % 0-centered frequency range, n-modes
fpoloidal = [-2,-1,0,1]; %m-modes

% B(:,1) = 0;
% B(:,:) = (fft2(b(:,:)));
% ftoroidal = 0:15;
% temp = [-16:-1];
% ftoroidal(1,(end+1):(end+length(temp))) = temp;



switch plotTypeString
    case 'FreqModesMesh'
        [az,el] = view;
        [X,Y] = meshgrid(ftoroidal,fpoloidal); % 0-centered 2D frequency
            range
        mesh(X,Y,abs(B))
        axis([-16 15 -2 1 0 30])
        view([az,el]);

    case
        {'ColumnShape','ColumnShapeMesh','ColumnShapeRed','CrossSection'}
        %% Going from Cmn (magnetic field) to Smn (magetic fieldlines
            displacment)

        Bsize = size(B);
```

```
        Smn = zeros(Bsize);

        Btoroidal = simData.btor_tfc;   %[mT]
%         Btoroidal = -9.627087041735649;

        mIndex = 0;
        for m = fpoloidal
            mIndex = mIndex +1;
            nIndex = 0;
            for n = ftoroidal
                nIndex = nIndex + 1;
                if n ~= 0 || m ~= 0
                    %when both modes are 0 this gives inf for Smn, not
                        good, ask why it is like this? remember:
                        Smn(mIndex,nIndex,tStep) =
                        B(mIndex,nIndex,N)/(1j*(m*BpoloidalN/r+n*BtoroidalN/R));
%                     mIndex = m-fpoloidal(1)+1;
%                     nIndex = n-ftoroidal(1)+1;

                    Smn(mIndex,nIndex) =
                        B(mIndex,nIndex)/(1j*(m*Bpoloidal/simData.aminor+n*Btoroidal/simData.Rmajor));
                end
             end
        end


        %% Going from Smn (magetic fieldlines displacment) to r (Plasma
            displacement)

        % simData.theta_res is a number which can be a fraction and
        % therefor needs to be rounded to prevent inconsistencies in the
            plot.
        thetaRes =
            -pi:(2*pi/round(simData.theta_res)):(pi-(2*pi/round(simData.theta_res)));
        phiRes =
            -pi:(2*pi/round(simData.phi_res)):(pi-(2*pi/round(simData.phi_res)));

        r = zeros(length(thetaRes),length(phiRes));

        SmnR = real(Smn);
        SmnIm = imag(Smn);

%         %test
%          Am = 2*abs(Smn);
%          alpha = atan2(SmnIm,SmnR);
%          ftoroidal2 = 0:15;

        row = 0;
        for theta = thetaRes
            row = row + 1;
             col = 0;
            for phi = phiRes
                col = col + 1;
                mIndex = 0;
```

```
                for m = fpoloidal
                    mIndex = mIndex +1;
                    nIndex = 0;
                    for n = ftoroidal
                            nIndex = nIndex +1;

%                             mIndex = m-fpoloidal(1)+1;
%                             nIndex = n-ftoroidal(1)+1;

                            temp = m*theta+n*phi;
                            EulerR = cos(temp);
                            EulerIm = sin(temp);

                            %test
%                            r(row,col) =
    r(row,col)+Am(4,nIndex)*cos(alpha(4,nIndex)+temp);


                            r(row,col) = r(row,col) +
                                SmnR(mIndex,nIndex)*EulerR-SmnIm(mIndex,nIndex)*EulerIm;

                            %+(SmnR(mIndex,nIndex)*EulerIm+SmnIm(mIndex,nIndex)*EulerR)*1j;
                            %im part
%                             (SmnR*EulerIm+SmnIm*EulerR);

                    end
                end
            end
        end

        %% Animation

        % simData.theta_res is a number which can be a fraction and
        % therefor needs to be rounded to prevent inconsistencies in the
            plot.
        thetaRes = -pi:(2*pi/round(simData.theta_res)):pi;
        phiRes = -pi:(2*pi/round(simData.phi_res)):pi;

        %Close the 2D-loop for animation
        rSize = size(r)+[1 1];
        rClosed = zeros(rSize);
        rClosed(1:(end-1),1:(end-1)) = r;
        rClosed(1:(end-1),end) = r(:,1);
        rClosed(end,(1:(end-1))) = r(1,:);

        %Boost amp to see disturbances better
        rClosed = rClosed*simData.amp_boost;

        x = zeros(size(r(:,:)));
        y = zeros(size(x));
        z = zeros(size(x));

        %unvectorized verison
        row = 0;
```

```
        for theta = thetaRes
            row = row + 1;
             col = 0;
            for phi = phiRes
                col = col + 1;

                x(row,col) = (simData.Rmajor + (simData.aminor +
                    rClosed(row,col)/1000) *cos(theta)) * cos(phi);
                y(row,col) = (simData.Rmajor + (simData.aminor +
                    rClosed(row,col)/1000) * cos(theta)) * sin(phi);
                z(row,col) = (simData.aminor + rClosed(row,col)/1000)*
                    sin(theta);

            end

        end
            if strcmp(plotTypeString,'CrossSection')
%               [az,el] = view;
%               plot(x(:,1), z(:,1));
        thetaRes2 =
            -pi:(2*pi/round(simData.theta_res)):(pi-(2*pi/round(simData.theta_res)));
        temp = (r(:,30)*simData.amp_boost/1000+simData.aminor);
%         temp = ones(length(thetaRes2),1)*0.2;
           x = cos(thetaRes2)'.*temp;
           y = sin(thetaRes2)'.*temp;
                plot(x,y)
%                 axis([-.8 .8 -.8 .8])
                axis equal
%                 view([az,el]);
                drawnow

            elseif strcmp(plotTypeString,'ColumnShapeRed')
                [az,el] = view;
                surf(x, y, z,
                    r(:,:),'FaceColor','red','EdgeColor','none');
                camlight left;
                lighting phong
                axis([-1.8 1.8 -1.8 1.8 -.8 .8])
                view([az,el]);
                drawnow

            elseif strcmp(plotTypeString,'ColumnShapeMesh')
                [az,el] = view;
                mesh(x, y, z);
                axis([-1.8 1.8 -1.8 1.8 -.8 .8])
                view([az,el]);
                drawnow

            else
                [az,el] = view;
                surf(x, y, z, r(:,:));
                axis([-1.8 1.8 -1.8 1.8 -.8 .8])
                view([az,el]);
                drawnow
```

```
            end

        end
    end
```

Data selection code

```
function [ simDataOut ] = pickOutData( handles )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

simDataOut.currentTimeStep =
    handles.currentTimeStep+round(get(handles.time_res,'Value'));

% Sensor data (32*4 flux sensors) 10 000 data points
% Start time: -50 ms = 1 index
% End time:  962.7 ms = 1000 index
% Time step: 0.1 ms
% Max time currentTimeStep: 9619

% Sensor data (Current etc) 8 192 data points
% Start time: -12 ms
% End time:  151.8 ms
% Time step: 0.02 ms
% Max time currentTimeStep: 8 192

simDataOut.currentTimeIndex1 = (50-12)/0.1+1+handles.currentTimeStep;
    %Starts at -12 ms = index 381 with 0.1 ms time step
simDataOut.currentTimeIndex2 = 5*handles.currentTimeStep;
    %Starts at -12 ms = index 1 with 0.1 ms time step

simDataOut.brad_A = handles.brad_A(:,simDataOut.currentTimeIndex1);
simDataOut.brad_B = handles.brad_B(:,simDataOut.currentTimeIndex1);
simDataOut.plas_curr = handles.plas_curr(simDataOut.currentTimeIndex2);
simDataOut.btor_tfc = handles.btor_tfc(simDataOut.currentTimeIndex2);

%% Parameters
simDataOut.Rmajor = 1.24; %[m]
simDataOut.aminor = 0.183; %[m]
simDataOut.theta_res = get(handles.theta_res,'Value');
simDataOut.phi_res = get(handles.phi_res,'Value');
simDataOut.amp_boost = get(handles.amp_boost,'Value');

end
```