The plasma position calculation algorithm consists of several files, one for each step of the process. You should download each file and place all of them in the same directory. On the end of this page there is the code implemented which you can try before downloading the files. However using the files should be easier if you intend to use the code on a regular basis.

SdasStart.m - Creates a link to the SDAS server, returning a SDASClient object

bobines_isttok_Port03_p15d0.m - Contains information about the magnetic probes and ISTTOK

SdasLoadMagneticData.m - Gets magnetic signals from the database

SdasLoadIvertData.m - Gets vertical current signals from the database

SdasLoadIhorizData.m - Gets horizontal current signals from the database

SdasCreateTimeVector.m - Auxiliary function called by SdasLoadMagneticData.m which creates the time vector of the signals

RemoveOffset.m - Removes the offset from the magnetic signals

IntegralCorrection.m - Performs integral correction in the magnetic signals due to losses in the analog integrators

CorrectMagneticField.m - Corrects the influence of the equilibrium magnetic fields

hpol_espira.m - Auxliary file for CorrectMagneticField.m

hRZ_espira.m - Auxliary file for CorrectMagneticField.m

CalculatePlasmaPosition_SA.m - Calculates the plasma position using the magnetic signals and a lightweight algorithm

CalculatePlasmaPosition_CF1f_SD.m - Calculates the plasma position using the current filaments algorithm (steepest descent minimization) with one filament

CalculatePlasmaCurrent.m - Calculates the plasma current using the magnetic probes. Auxiliary file for CalculatePlasmaPosition_CF1f_SD.m

DspResampleData.m - Auxiliary file to resample some data (Ivert and Ihoriz)

1. Create a link to the SDAS server. Calling this function returns a SDAS Client object, named ***client***, which will be passed into the function that fetches data from the database. This needs to be called only once and only works if the server is on 'baco.ipfn.ist.utl.pt' and on port 8888:

```
client = SdasStart();
```

2. Load magnetic probes information, such as the position of the probes and information about the dimensions of ISTTOK. Once again this can be done only once:

```
magSensors = bobines_isttok_Port03_p15d0
```

3. Get magnetic and current signals from the database, replace ***shotnr*** with the desired shot number:

```
[magTime rawData] = SdasLoadMagneticData(client, shotnr);
rawData = -rawData;  % It must be here for now (2008-01-15).
[ivertTime ivertData] = SdasLoadIvertData(client, shotnr);
[ihorizTime ihorizData] = SdasLoadIhorizData(client, shotnr);
```

4. Resample Ivert and Ihoriz data:

```
ivertResampledData = DspResampleData(ivertTime, ivertData, magTime);
ihorizResampledData = DspResampleData(ihorizTime, ihorizData, magTime);
```

5. Remove the DC offset from the magnetic signals, where ***100*** represents the number of initial samples used to calculate the offset:

```
Hp_NoOffset = RemoveOffset(rawData, 100);
```

6. Correct the magnetic signals due to the analog integrators:

```
Hp = IntegralCorrection(Hp_NoOffset);
```

7. Correct the equilibrium field influence:

```
Hpc = CorrectMagneticField(Hp, ivertResampledData, ihorizResampledData,
magSensors);
```

8. Calculate the plasma position (lightweight algorithm):

```
[R Z] = CalculatePlasmaPosition_SA(Hpc, magSensors);
```

or (current filaments algorithm, very slow)

```
[R Z] = CalculatePlasmaPosition_CF1f_SD(Hpc, magSensors);
```

9. Plot the results:

```
%% Plot position
figure;
plot(magTime/1000, R, 'r', magTime/1000, Z, 'b');
legend('R', 'Z');
xlabel('Time (ms)');
ylabel('Position (m)');
grid on;
```

This is an overview of the implemented code. You can use this for testing purposes before downloading the above files. Just copy each section, paste it directly into the Matlab console and execute it. It doesn't use any special library and if SDAS was correctly installed it should work.

1. Get magnetic signals from the database (with SDAS and don't forget to modify shotnr to the shot you want)

```
%% Necessary Java imports
import org.sdas.core.client.*;
import org.sdas.core.time.*;
import java.lang.*;

%% Shot number
shotnr = 16279;

%% SDAS Client
client = SDASClient('baco.ipfn.ist.utl.pt', 8888);

%% Complete UID for the parameters (shot dependent)
if shotnr < 15669
  parameterUid(1,:)  = 'TR512_TOMOGRAPHY.TR512_B04.CHANNEL_0';
  parameterUid(2,:)  = 'TR512_TOMOGRAPHY._TR512_B04.CHANNEL_1';
  parameterUid(3,:)  = 'TR512_TOMOGRAPHY.TR512_B04.CHANNEL_2';
  parameterUid(4,:)  = 'TR512_TOMOGRAPHY.TR512_B04.CHANNEL_3';
  parameterUid(5,:)  = 'TR512_TOMOGRAPHY.TR512_B04.CHANNEL_4';
  parameterUid(6,:)  = 'TR512_TOMOGRAPHY.TR512_B04.CHANNEL_5';
  parameterUid(7,:)  = 'TR512_TOMOGRAPHY.TR512_B04.CHANNEL_6';
  parameterUid(8,:)  = 'TR512_TOMOGRAPHY.TR512_B04.CHANNEL_7';
  parameterUid(9,:)  = 'TR512_TOMOGRAPHY.TR512_B03.CHANNEL_0';
  parameterUid(10,:) = 'TR512_TOMOGRAPHY.TR512_B03.CHANNEL_1';
  parameterUid(11,:) = 'TR512_TOMOGRAPHY.TR512_B03.CHANNEL_2';
  parameterUid(12,:) = 'TR512_TOMOGRAPHY.TR512_B03.CHANNEL_3';
end
if shotnr >= 15669
  parameterUid(1,:)  = 'PLASMACONTROL.TR512_B00.CHANNEL_0';
  parameterUid(2,:)  = 'PLASMACONTROL.TR512_B00.CHANNEL_1';
  parameterUid(3,:)  = 'PLASMACONTROL.TR512_B01.CHANNEL_0';
  parameterUid(4,:)  = 'PLASMACONTROL.TR512_B00.CHANNEL_2';
  parameterUid(5,:)  = 'PLASMACONTROL.TR512_B00.CHANNEL_3';
  parameterUid(6,:)  = 'PLASMACONTROL.TR512_B01.CHANNEL_1';
  parameterUid(7,:)  = 'PLASMACONTROL.TR512_B00.CHANNEL_4';
  parameterUid(8,:)  = 'PLASMACONTROL.TR512_B00.CHANNEL_5';
  parameterUid(9,:)  = 'PLASMACONTROL.TR512_B01.CHANNEL_2';
  parameterUid(10,:) = 'PLASMACONTROL.TR512_B00.CHANNEL_6';
  parameterUid(11,:) = 'PLASMACONTROL.TR512_B00.CHANNEL_7';
  parameterUid(12,:) = 'PLASMACONTROL.TR512_B01.CHANNEL_3';
end
```

```
%% Search existing parameters
parameterList = '';
for i = 1:12
  if client.parameterExists(parameterUid(i,:), '0x0000', shotnr)
    parameterList = strvcat(parameterList, parameterUid(i,:));
  end
end

%% Convert names to Java Strings
parameterListCell = javaArray('java.lang.String', 1, size(parameterList, 
1));
for i = 1:size(parameterList, 1)
  parameterListCell(1, i) = String(parameterList(i,:));
  parameterListCell(1, i) = parameterListCell(1, i).trim();
end

%% Get magnetic data
parameterDataStruct = client.getMultipleData(cell(parameterListCell), 
'0x0000', shotnr);
DummyData = parameterDataStruct(1).getData();
magneticData = zeros(size(DummyData, 1), length(parameterDataStruct));
for i = 1:length(parameterDataStruct)
  magneticData(:,i) = parameterDataStruct(i).getData();
end
magneticData = double(magneticData);

%% Create time vector
% Get parameter start time, end time and time between samples
data = parameterDataStruct(1).getData();
tStart = parameterDataStruct(1).getTStart().getTimeInMicros();
tEnd = parameterDataStruct(1).getTEnd().getTimeInMicros();
tbs = (tEnd - tStart) / length(data);
% Get event time and determine the data initial delay
Events = parameterDataStruct(1).getEvents();
tEvent = Events(1).getTimeStamp().getTimeInMicros();
Delay = tStart - tEvent;
% Time vector
Time = (Delay:tbs:(Delay + (length(data) - 1) * tbs))';
```

2. Create magnetic sensor information

```
%% Basic sensor information
N  = 12;               % Number of probes
RM = .46;              % Major radius
rb = .187/2;           % Probe array radius
a  = 0.085;            % Minor radius

%% Probe positions
n=0:N-1;
```

```
theta0 = 0;
R  = rb * cos(theta0 / 180 * pi - n/6 * pi) + RM;
Z  = rb * sin(theta0 / 180 * pi - n/6 * pi);

%% Probe direction (versors)
epR =  sin(theta0 / 180 * pi - n*pi/6);
epZ = -cos(theta0 / 180 * pi - n*pi/6);

%% Structure containing sensor information
magsensors.RM  = RM;
magsensors.rb  = rb;
magsensors.a   = a;
magsensors.R   = R;
magsensors.Z   = Z;
magsensors.epR = epR;
magsensors.epZ = epZ;
```

3. Remove DC offset from signals

```
%% Calculate the offset
offset = mean(magneticData(1:size(magneticData, 1),:));
offsetMatrix = repmat(offset, size(magneticData, 1), 1);

%% Remove the offset
magData = magneticData - offsetMatrix;
```

4. Correct the signals due to the analog integration

```
%% Integral correction constant
IntegralCorrectionConstant = 10.6383 / 2000000.0;

%% Calculate cumulative sum
dataCumSum = cumsum(magData);

%% Correct data
correctedData = magData + IntegralCorrectionConstant * dataCumSum;
```

5. Calculate the plasma position

```
%% Initialize some variables
R = zeros(size(correctedData, 1), 1);
Z = zeros(size(correctedData, 1), 1);
magneticDataSum = sum(correctedData, 2);
Rb = repmat(magsensors.R, size(correctedData, 1), 1);
Zb = repmat(magsensors.Z, size(correctedData, 1), 1);
productMagneticR = correctedData .* Rb;
productMagneticZ = correctedData .* Zb;
```

```
sumProductMagneticR = sum(productMagneticR, 2);
sumProductMagneticZ = sum(productMagneticZ, 2);

%% Plasma position - simple algorithm
%  Iterate all the samples
for i = 1:size(correctedData, 1)
  % Calculate plasma position
  R(i) = 2*(sumProductMagneticR(i) / magneticDataSum(i) - magsensors.RM)
+ 0.04 - 0.006;
  Z(i) = 2*(sumProductMagneticZ(i) / magneticDataSum(i));
  % Saturate plasma position
  if R(i) < -magsensors.a
    R(i) = -magsensors.a;
  end
  if R(i) > magsensors.a
    R(i) = magsensors.a;
  end
  if Z(i) < -magsensors.a
    Z(i) = -magsensors.a;
  end
  if Z(i) > magsensors.a
    Z(i) = magsensors.a;
  end
end
```

6. Plot plasma position

```
%% Plot position
figure;
plot(Time/1000, R, 'r', Time/1000, Z, 'b');
legend('R', 'Z');
xlabel('Time (ms)');
ylabel('Position (m)');
grid on;
```