**Infrared beam monitoring of XFEL injector laser**

Louis De Gruy, University of Dayton, United States of America

Supervisor: Florian Kaiser

Co-Supervisor: Michael Schulz

11 September 2014

**Abstract**

This report examines the process of creating and maintaining a reliable system to be used to monitor the infrared pulse characteristics of the European XFEL injector laser. The system involves using a single-shot autocorrelator and a custom MATLAB program to capture and process images of the laser beam. Although the injection laser itself was not fully operational at the time of writing, the characterization system was determined to perform well when used with a beam with comparable pulse characteristics.

1. **Introduction**

The European X-ray free electron laser (XFEL) project is an international project with 12 collaborating countries that aims to provide a world-class X-ray laser research facility located at the Deutsches Elektronen-Synchrotron (DESY) in Hamburg, Germany. The project consists of a free-electron laser generating high-intensity electromagnetic radiation by accelerating electrons to relativistic speeds and producing X-ray pulses in synchronization. These signals have the properties of laser light and are much more intense than those produced by conventional means. Additionally, the pulse durations of these signals will be less than a few femtoseconds, allowing for the observation of chemical interactions that are too rapid to be observed by other methods.1 The facility hopes to further research in a wide range of disciplines, including physics, chemistry, materials, science, biology, and nanotechnology.

The FL-SA workgroup at DESY is responsible for the development, construction, and maintenance of the injection laser at XFEL. Responsible for the creation of the electrons used in the XFEL, the injection laser is a critical component of the project, and is required to meet specific parameters. To ensure that these parameters are met, the beam of the injection laser itself must be characterized and monitored carefully to ensure a reliable source of electrons for the project. However, because the injector laser operates at pulses faster than what can be measured by conventional methods, an autocorrelation system was developed to effectively characterize and monitor the beam.

1. **Theory**

Although many conventional detectors exist which are capable of well-defined temporal resolution, these still do not have the resolution capabilities to detect an ultrashort laser pulse (that is, pulse durations on the order of pico- or femtoseconds). In response to this, various methods of ultrashort pulse detection have been developed. One such method is called second harmonic autocorrelation. The general principle of SH autocorrelation involves splitting a laser beam into two equal-intensity beams and recombining them in a nonlinear crystal. One beam passes through a delay stage, which allows for user-defined time delay, *td*. The beam overlap within the crystal produces second harmonic generation. Because second harmonic intensities are proportional to the product of the intensities of the two beams (for the low-pump depletion regime), the measurement of SH as a function of relative time delay between the two beams provides the autocorrelation function *A(td)*, which goes as

|  |  |
| --- | --- |
|  | (1) |

Where *I(t)* represents the intensities of the two beams as a function of time *t*. If the pulse shape of the beam is assumed, the pulse duration can be determined from the full width at half maximum of the autocorrelation function. In a multi-shot scanning operation, the AC function can be measured by introducing different time-delays in the system.

However, in order to measure pulses of durations from tens of femtoseconds to a few picoseconds, a passive, more efficient setup can be used that makes active delay scanning unnecessary. Such a method allows for single-shot laser pulse characterization. Similar to the multi-shot technique, a laser beam is split into two beams of equal intensity, and the two beams are made to cross at a (small) angle inside a non-linear crystal. As shown in Figure 1, the two beams can be made to overlap in space and time, provided that the beam waist in the overlap is significantly larger than the pulse length. With this geometry, the SH signal generated at each overlapping point is proportional to the product of the intensities of each beam paths. The direction of this emitted SH signal propagates in the same direction as the bisector of the beam intersection angle.

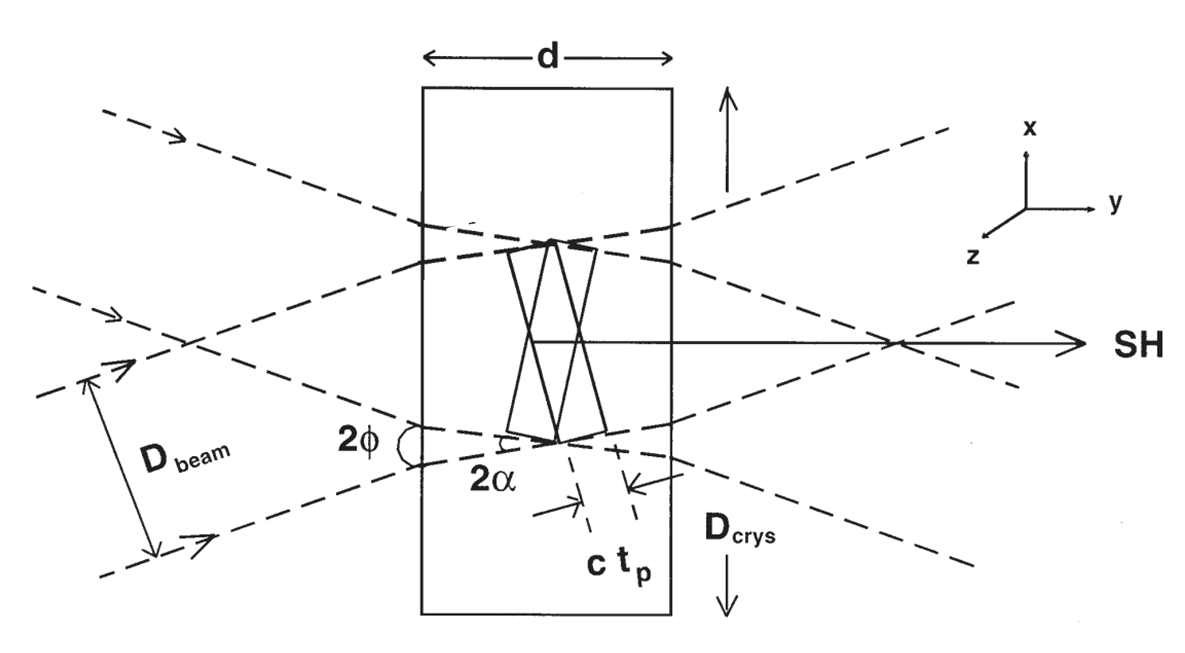


Figure 1: Two-beam temporal overlap inside a non-linear crystal. *D*beam represents the beam diameter of the incoming beams, 2φ is the beam crossover angle on the exterior of the crystal, 2α is the crossover angle inside of the crystal, *ctp* is the spatial pulse length, *d* is the thickness of the crystal, *D*crys is the crystal aperture, and SH designates the direction of the second harmonic generated frequency.

A result is that, for any line parallel to the bisector, the generated SH signal represents the intensity product integrated over time. As a result, different lines parallel to the bisector designate overlaps for different time delays. The line made by the bisector of the overlap region, the central line, designates the full overlap of the two pulses. In fact, for time delays that deviate from that which provides the maximum intensity, the beam center will move along the *x*-direction. Because of this, the autocorrelation function can now be represented as a spatial intensity distribution along the *x* direction. This intensity distribution can be recorded with a CCD camera.

Important components to consider in a single-shot autocorrelator are the physical size of the nonlinear crystal, the orientation of the crystal, the diameter of the overlapping beams, the expected pulse duration, and the angle between the beams.­1

Referring to Figure 1, the necessary conditions on the beam diameter and crystal thickness can be obtained. For beams having diameter *D*beam, a complete temporal overlap occurs when

|  |  |
| --- | --- |
| . | (2) |

And, for a crystal thickness of *d*, temporal overlap occurs when

|  |  |
| --- | --- |
| . | (3) |

Where *c* is the speed of light in a vacuum, *n* is the refractive index of the crystal and *tp*is the pulse duration of the laser beam. Given the exterior overlap angle φ and the refractive index of the crystal, the interior overlap angle α can be calculated using Snell’s Law.

Additionally, the crystal aperture, *D*crys, must be large enough to allow both beams to pass through without being clipped. As a result,

|  |  |
| --- | --- |
| . | (4) |

1. **Experiment**

Since the goal of this experiment is to characterize pulses up to 2ps at 1030nm, the following parameters were selected.

A BBO crystal was selected for use as the non-linear crystal in this autocorrelator setup. It had a thickness *d*=0.5mm, an aperture width *D*crys=5mm, and was cut at 23.4° to maximize SH generation at 1030nm through birefringent phase matching.

For convenience, we selected a reasonably sized *D*beam of 2mm and an overlap angle that yields φ=19°, (therefore, α=11.3°). Using equations 2 through 4, we find that our setup can handle pulse durations up to *tp, max=* 2.29 ps.

Figures 2 and 3 depict the experimental setup for the single-shot autocorrelator. This autocorrelator uses the principles of noncollinear geometry in order to produce the second harmonic generation of the beam to be characterized.

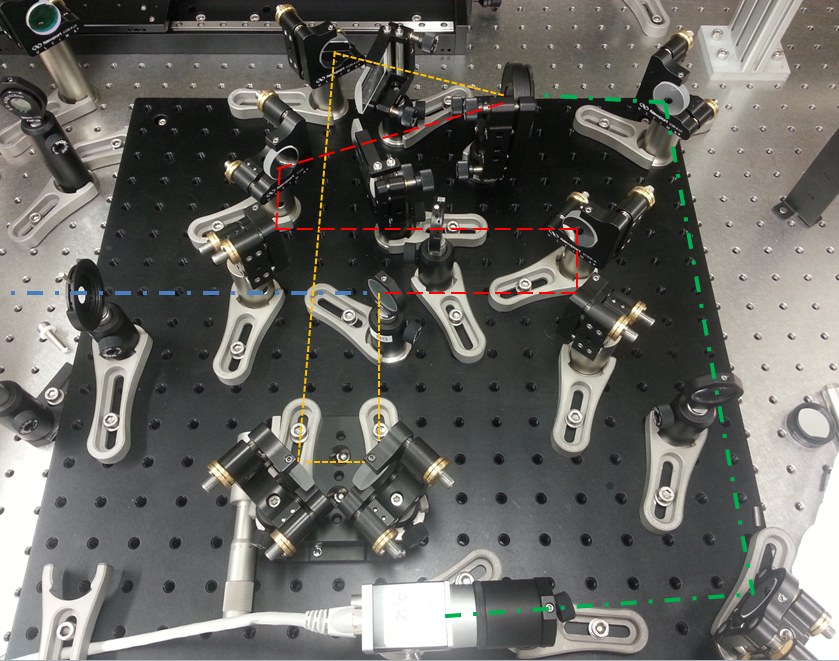


Figure 2: The autocorrelator setup, with colored lines representing beam paths.

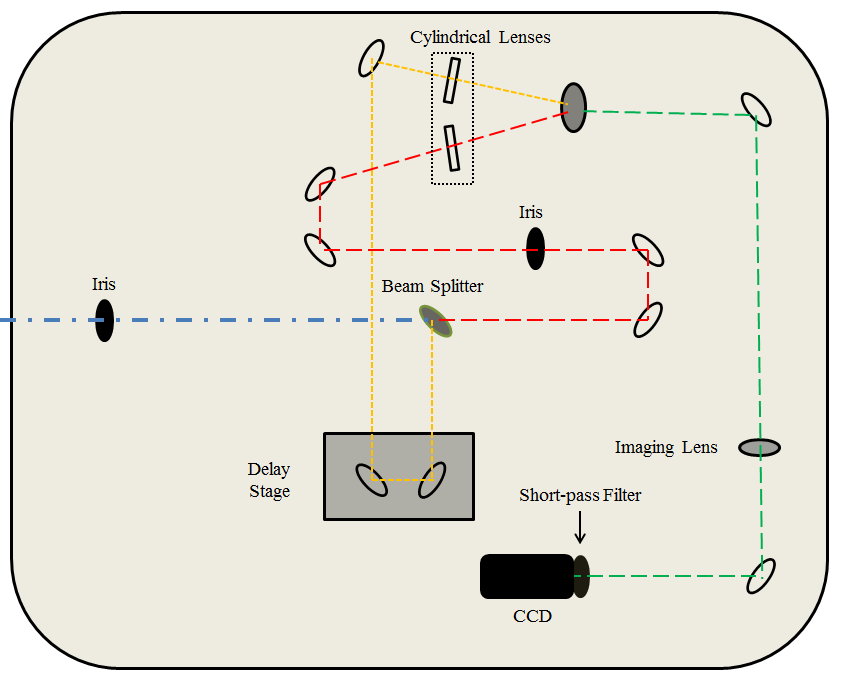


Figure 3: A simpllified overhead view of the autocorrelator. Unlabeled components represent path mirrors. The depicted irises were installed for maintenance alignment of the beam.

Because the injector laser was not fully operational at the time of publication, another laser (the OneFive Origami-10) of similar pulse characteristics was used. The laser produced a beam of λ=1030nm, *f*=108MHz, average power: 200mW, power per pulse: 1.8nJ, and pulse length: 230fs.

This beam was split into two equal-intensity beams using a 50/50 beamsplitter. Both beams must follow equidistant paths before arriving and overlapping inside the BBO crystal. To accomplish this, one beam (orange) travels via two mirrors on a delay stage (with attached micrometer screw), which is then used to account for minute differences in path length.

In order to increase beam intensity without sacrificing beam width, the beams were guided through line-focusing cylindrical lenses before entering the BBO crystal. As the beams overlap inside the BBO, second harmonic generation occurs, and the SH signal (green) is emitted through the crystal. An imaging lens with *f*=150mm is used to image the signal from the BBO into a CCD camera using a 2*f*-2*f* configuration. A 515nm band-pass filter then removes stray light at 1030nm.

1. **Results and Characterization**

Capturing beam images using this setup as described does not, in itself, fully characterize the beam. It is then necessary to calibrate the device in order to provide an understanding of the beam itself. Figure 4 shows that there is, as the theory section of this report suggests, a deviation in beam center position and a change in beam intensity when a time-delay is introduced. However, the images themselves do not easily describe the specific values of the beam center positions or the intensity changes.

In order to determine both of these changes in value, a sample of 19 images was taken at regular time delay intervals such that -2000fs ≤ *td* ≤ 2000fs. These images were then treated using the ColumnSum program in MATLAB to determine the amplitude of the beam for that specific time delay. For a description of ColumnSum and how it was used to obtain the amplitudes of the beam for the different time delays, please see Appendix 1. A sample of images with their ColumnSum treatment can be found in Figure 4. Additionally, a plot of the 19 beam intensities versus time delay can be found in Figure 5.

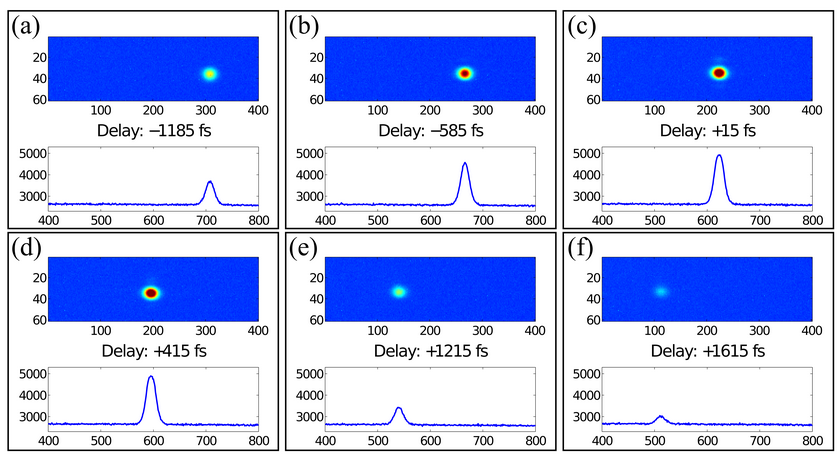


Figure 4: As predicted by the theory section of this report, introducing a time delay in the arrival of a single beam’s pulse causes beam intensity to decrease and the beam center to shift in the *x*-direction. The vertical axis of the plots accompanying each image represents beam intensity, and the horizontal axis depicts beam position in terms of pixels on the CCD used.

Further inspection of the sample images in Figure 4 confirms another theory described earlier in the report: the beam center shifts along the *x*-direction when a time delay is introduced. To determine the exact relationship between time delay and beam center shift, a plot of center position v. time delay was made. Inspection of the plot shows the relationship to be linear, with a slope (describing change in beam center position in pixels per 1fs delay) of -0.0695 pixel/fs (i.e. 14.38 fs/pixel). This slope provides a calibration factor which we can use to determine the beam’s real position in space. The CCD we used, Basler A1300-60gm NIR, is such that a single pixel in an image represents 5.3m in space. Therefore, the actual change in beam position v. time delay is 368nm/fs.

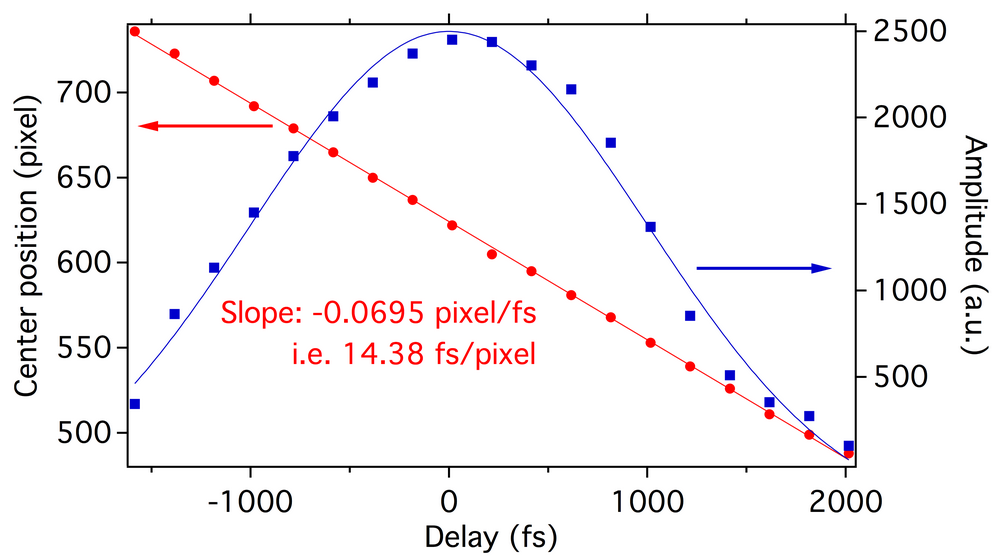


Figure 5: A summary of images taken from the autocorrelator-CCD setup. The beam amplitude v. time delay plot is printed in blue, and utilizes the vertical axis on the right. The beam center v. time delay is printed in red, and utilizes the vertical axis on the left.

Figure 5 shows that this autocorrelator can only be used for pulse durations at around 2000fs (2ps), where the amplitude of the beam is reasonably high. This was predicted in the Theory section of this report.

For the sample size of images collected, the average FWHM (a measurement of pulse length) was 225ps, which is on par with the measurement of 230fs taken by an A.P.E. PulseCheck 50 commercial autocorrelator. Therefore, we can conclude that this autocorrelator is an acceptable tool in measuring characteristics of the injection laser.

1. **Developing a GUI for use in Data Acquisition**

The core program used to capture the images from the autocorrelator-CCD configuration is not, necessarily, user-friendly. To assist in the treatment and analysis of the captured images, a GUI was developed using MATLAB. The primary function of the code consists of a while loop that initiates at the users command, and terminates also at the user’s command. The loop is responsible for capturing the image, filtering the image for noise, plotting the amplitude, radius, and center of the beam, and applying a Gaussian fit to each image. Additionally, after the first image is captured, each following image is cropped according to user inputs given at the initialization of the program. After the user terminates the loop, the user is then given the option to save the captured images. The full MATLAB code can be found in Appendix 2.

1. **Suggested Improvements**

The primary deficiencies of this setup lie in the software used for image acquisition. Certain time-consuming processes take place inside of the iterative loop, which severely hampers the overall rate of image acquisition. Additionally, the GUI does not currently allow for the user to change image parameters while images are being acquired and processed. As it stands now, the user must restart the GUI in order to change the image parameters.

1. **REFERENCES**
2. “XFEL Facts and Figures”; http://www.xfel.eu/overview/facts\_and\_figures/
3. Raghuramaiah, Sharma, Naik, Gupta, Ganeev. 2001, “A second-order autocorrelator for single-shot measurement of femtosecond laser pulse durations.” *Sadhana*, Vol. 26, Part 6, pp 603-611.
4. Tautz. 2008, ”Single-shot characterization of sub-10fs laser pulses.”

**APPENDIX**

1. **A description of ColumnSum and how it was used in this report**

To understand ColumnSum, one must first understand that MATLAB processes and stores images as arrays, with integer values stored in each element that correspond to the intensity of light at that particular pixel of the image. To calculate the maximum amplitude of each image, Columnsum was used to sum the intensities of each column of pixels. These sums were then stored and plotted similar to those used in the report.

1. **Full MATLAB GUI code**

function varargout = serious(varargin)  
% SERIOUS MATLAB code for serious.fig  
%      SERIOUS, by itself, creates a new SERIOUS or raises the existing  
%      singleton\*.  
%      H = SERIOUS returns the handle to a new SERIOUS or the handle to  
%      the existing singleton\*.  
%      SERIOUS('CALLBACK',hObject,eventData,handles,...) calls the local  
%      function named CALLBACK in SERIOUS.M with the given input arguments.  
%      SERIOUS('Property','Value',...) creates a new SERIOUS or raises the  
%      existing singleton\*.  Starting from the left, property value pairs are  
%      applied to the GUI before serious\_OpeningFcn gets called.  An  
%      unrecognized property name or invalid value makes property application  
%      stop.  All inputs are passed to serious\_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 serious  
  
% Last Modified by GUIDE v2.5 29-Aug-2014 09:23:08  
warning('off','all');  
% Begin initialization code - DO NOT EDIT  
gui\_Singleton = 1;  
gui\_State = struct('gui\_Name',       mfilename, ...  
                   'gui\_Singleton',  gui\_Singleton, ...  
                   'gui\_OpeningFcn', @serious\_OpeningFcn, ...  
                   'gui\_OutputFcn',  @serious\_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 serious is made visible.  
function serious\_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 serious (see VARARGIN)  
  
% Choose default command line output for serious  
handles.output = hObject;  
NoiseFilter=0;    %Do you want to filter noise in the image? 1=yes, 0=no  
first=1;  
  
PixelSize=4.4e-3; %Pixel size (mm)  
  
prompt = {'Enter Y Pixel Start:','Enter Y Pixel Width:', 'Enter Plot Width', 'Enter Smoothing Points'};  
dlg\_title = 'Input';  
num\_lines = 1;  
def = {'','','',''};  
answer = inputdlg(prompt,dlg\_title,num\_lines,def);  
display(answer);  
  
xPixelsStart=1;  
xPixelsWidth=1279;  
  
%Define handles  
handles.first=first;  
handles.Noise=1;  
handles.pixsize=PixelSize;  
handles.xPS=xPixelsStart;  
handles.xPW=xPixelsWidth;  
handles.yPS=str2double(answer(1:1));  
handles.yPW=str2double(answer(2:1));  
handles.plotwidth=str2double(answer(3:1));  
handles.Smoothing=str2double(answer(4:1));  
  
% Choose default command line output for Test2  
handles.output = hObject;  
% Update handles structure  
guidata(hObject, handles);  
  
% UIWAIT makes serious wait for user response (see UIRESUME)  
% uiwait(handles.figure1);  
  
% --- Outputs from this function are returned to the command line.  
function varargout = serious\_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;  
  
% --- Executes on button press in refigure.  
function refigure\_Callback(hObject, eventdata, handles)  
% hObject    handle to refigure (see GCBO)  
% eventdata  reserved - to be defined in a future version of MATLAB  
% handles    structure with handles and user data (see GUIDATA)  
i=0;

while get(hObject, 'Value')  
     
[CorrImage, imerror] = jdoocs\_call('XFEL.DIAG/CAM.XFELCPULASER1/IRAutoCorrelator/IMAGE\_EXT');  
if isempty(imerror)  
else  
    warning('Error retrieving image from Nearfield camera')  
end  
%Crop all images after the first CamImg  
if handles.first  
   CroppedImage=CorrImage;  
   handles.first=0;  
else  
     CroppedImage = imcrop(CorrImage,[handles.xPS handles.yPS handles.xPW handles.yPW]);   %Cut out the region of interest  
end  
%Filter  
if handles.Noise  
    WienerImage  = wiener2(CroppedImage,[3 3]);  
else  
    WienerImage = CroppedImage;  
end  
handles.camera1=WienerImage;  
axes(handles.CamImg);  
imagesc(handles.camera1);  
  
GaussFit = fittype('A\*exp(-((x-xc)/b)^2)+y0');  
options = fitoptions(GaussFit);  
options.Lower = [100 10 100 0];   
options.Upper = [1e5 100 1180  1e5];  
  
x=1;  
i=i+1;  
  
ColumnSum = sum(WienerImage);  
[rows, columns] = size(ColumnSum);  
xPosition = (handles.xPS:1:(handles.xPS+handles.xPW));  
  
ActualFit = fit(xPosition',ColumnSum',GaussFit,options);  
CoefficientList(i,:) = coeffvalues(ActualFit);  
  
handles.FitCurve = CoefficientList(i,1)\*exp(-((xPosition-CoefficientList(i,3))/CoefficientList(i,2)).^2)+CoefficientList(i,4);  
handles.xPS = round(CoefficientList(i,3)-3\*CoefficientList(i,2));  
handles.xPW = round(6\*CoefficientList(i,2));  
Repetition(i,1) = i;

PlotEnd = i;  
PlotStart = i - handles.plotwidth;  
if PlotStart<1  
    PlotStart = 1;  
else  
end  
axes(handles.axes2)  
 plot(Repetition(PlotStart:PlotEnd,1),CoefficientList(PlotStart:PlotEnd,1),Repetition(PlotStart:PlotEnd,1),smooth(CoefficientList(PlotStart:PlotEnd,1),4))

plot(Repetition(PlotStart:PlotEnd,1),CoefficientList(PlotStart:PlotEnd,1),Repetition(PlotStart:PlotEnd,1),smooth(CoefficientList(PlotStart:PlotEnd,1),handles.Smoothing))  
xlabel('Repetition #')  
ylabel('Amplitude (a.u)')  
  
axes(handles.axes3)  
plot(Repetition(PlotStart:PlotEnd,1),CoefficientList(PlotStart:PlotEnd,2),Repetition(PlotStart:PlotEnd,1),smooth(CoefficientList(PlotStart:PlotEnd,2),handles.Smoothing))  
xlabel('Repetition #')  
ylabel('1/e^2 radius (mm)')  
  
axes(handles.axes4)  
plot(Repetition(PlotStart:PlotEnd,1),CoefficientList(PlotStart:PlotEnd,3),Repetition(PlotStart:PlotEnd,1),smooth(CoefficientList(PlotStart:PlotEnd,3),handles.Smoothing))  
xlabel('Repetition #')  
ylabel('Beam Center (mm)')  
  
axes(handles.axes5)  
imagesc(WienerImage)  
  
axes(handles.axes6)  
plot(xPosition,ColumnSum,xPosition,handles.FitCurve)  
  
guidata(hObject,handles)  
end  
  
% --- Executes on selection change in popupmenu1.  
function popupmenu1\_Callback(hObject, eventdata, handles)  
% hObject    handle to popupmenu1 (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 popupmenu1 contents as cell array  
%        contents{get(hObject,'Value')} returns selected item from popupmenu1  
  
% --- Executes during object creation, after setting all properties.  
function popupmenu1\_CreateFcn(hObject, eventdata, handles)  
% hObject    handle to popupmenu1 (see GCBO)  
% eventdata  reserved - to be defined in a future version of MATLAB  
% handles    empty - handles not created until after all CreateFcns called  
%[CorrImage] = jdoocs\_call('XFEL.DIAG/CAM.XFELCPULASER1/IRAutoCorrelator/IMAGE\_EXT');  
% 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 NoiseFilter.  
function NoiseFilter\_Callback(hObject, eventdata, handles)  
% hObject    handle to NoiseFilter (see GCBO)  
% eventdata  reserved - to be defined in a future version of MATLAB  
% handles    structure with handles and user data (see GUIDATA)  
if get(hObject,'Value')  
    handles.Noise=1;  
else  
    handles.Noise=0;  
end  
guidata(hObject,handles)  
% Hint: get(hObject,'Value') returns toggle state of NoiseFilter  
  
  
% --- Executes on button press in usersave.  
function usersave\_Callback(hObject, eventdata, handles)  
% hObject    handle to usersave (see GCBO)  
% eventdata  reserved - to be defined in a future version of MATLAB  
% handles    structure with handles and user data (see GUIDATA)