AN IMAGING SYSTEM FOR MONITORING AN ULTRACOLD CLOUD OF ATOMS

Submitted by

Debanuj Chatterjee

IISER Kolkata

IISER KOLKATA

Supervised by

Dr. Sebastian Hofferberth

University of Stuttgart

Universität Stuttgart

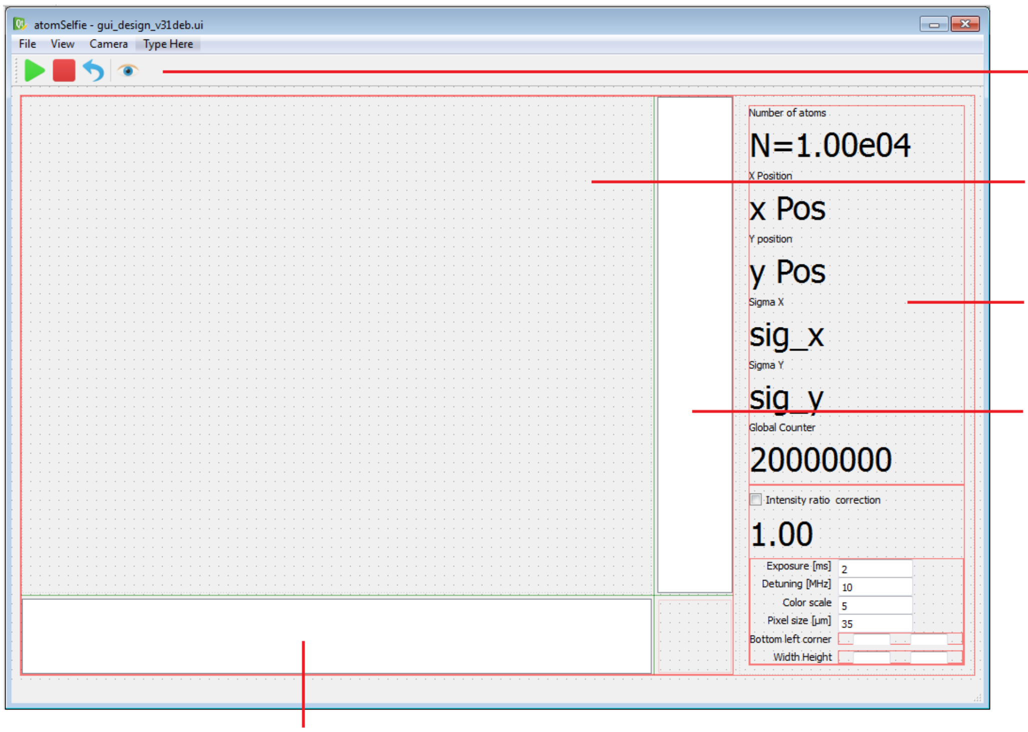
Introduction

An imaging system with a suitable graphical user interface(GUI) was developed using Python and Qt, for the study of an ultracold cloud of atoms. With the help of image processing and other fitting modules we could successfully determine the real-time position, size and the number of atoms in the cloud inside the experiment chamber. The intensity ratio correction was added to avoid the error due to the intensity fluctuations of the imaging beam. The camera trigger was also interfaced with the global counter of the setup and after successful image acquisition the compressed images were stored in the server.

Features of the imaging system

- **GUI Layout:**

The GUI for the system was developed using Qt4. A grid layout was set inside the main window and subsequently elements were added inside the grid.



Number of atoms
N=1.00e04
X Position
x Pos
Y position
y Pos
Sigma X
sig_x
Sigma Y
sig_y
Global Counter
20000000
□ Intensity ratio correction
1.00
Exposure [ms] 2
Detuning [MHz] 10
Color scale 5
Pixel size [µm] 35
Bottom left corner
Width Height

Toolbar

Matplotlib container

Image parameters

Fitting function display

Fitting function display

Fig 1: Layout of the GUI.

As one can see from Fig 1, a major portion of the main window was allotted to the Matplotlib container for display of the images.

The display for the fitting functions were added around the Matplotlib container.

On the right we added a vertical layout which displays the calculated parameters for the image.

On the bottom right corner a form layout was added to set different imaging parameters and the input of coordinates for a dragable rectangle.

A toolbar was placed on the top of the main window.

- **Camera input :**

We have two different modes of input from the camera.

1. Dummy cam : We can acquire images from a program that mimics the original camera and generates a simulated image of a cloud of atoms. This is required for dry runs to test the program when the experiment is not running.
2. Pixelfly camera : We can acquire images from the original Pixelfly camera when the experiment is running and the camera is triggered.

- **Image display :**

We use Matplotlib for displaying the images in the main window. The camera is triggered to acquire three different images: atom with the imaging light(A), only the light(L) and the background(B). These three images are displayed in smaller panes on the right side of the Matplotlib container(see Fig 3).

In order to make the system work faster we reduce the size of these images by doing a coarse graining. This is nothing but skipping some pixels of the image while displaying. The amount of coarse graining can be changed from the configuration file.

Once these three images are acquired the final image(F) is produced by computing : F=log((L-B)/(A-B)). Then the final image is displayed on the larger pane of the Matplotlib container(see Fig 3).

The Matplotlib display comes with an interactive navigation toolbar. This toolbar provides us the following functionalities (see Fig 2):

1. Pan/Zoom button : This button has two modes: pan and zoom. We can click this button to activate panning and zooming, then mouse is put somewhere over an axes. The left mouse button can be pressed and holding it will pan the figure, dragging it to a new position. On release, the data under the point where we pressed will be moved to the point where we released. If we press ‘x’ or ‘y’ while panning the motion will be constrained to the x or y axis, respectively. Pressing the right mouse button will zoom, dragging it to a new position. The x axis will be zoomed in proportionate to the rightward movement and zoomed out proportionate to the leftward movement. Same for the y axis and up/down motions. The point under the mouse when we begin the zoom remains stationary, allowing us to zoom to an arbitrary point in the figure. We can use the modifier keys ‘x’, ‘y’ or ‘CONTROL’ to constrain the zoom to the x axis, the y axis, or preserve the aspect ratio, respectively.
2. Zoom-to-rectangle button : Clicking this button activates this functionality. We can put the mouse somewhere over an axes, press the left mouse button, drag the mouse while holding the

button to a new location and release. The axes view limits will be zoomed to the rectangle we have defined by the drag. There is also an experimental 'zoom out to rectangle' in this mode with the right button, which will place our entire axes in the region defined by the zoom out rectangle.

3. Save button : We can click this button to launch a file save dialog and save the images with the extensions: png, ps, eps, svg and pdf.
4. Home, forward and back button : The forward and back buttons are used to navigate back and forth between previously defined views. They have no meaning unless we have already navigated somewhere else using the pan and zoom buttons. This is analogous to trying to click Back on your web browser before visiting a new page –nothing happens. Home button always takes you to the first, default view of your data.

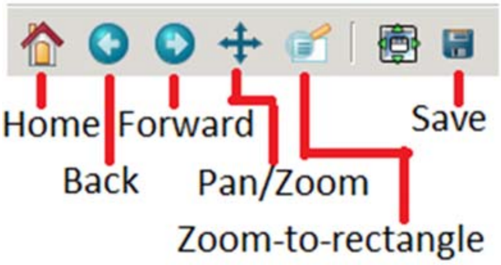
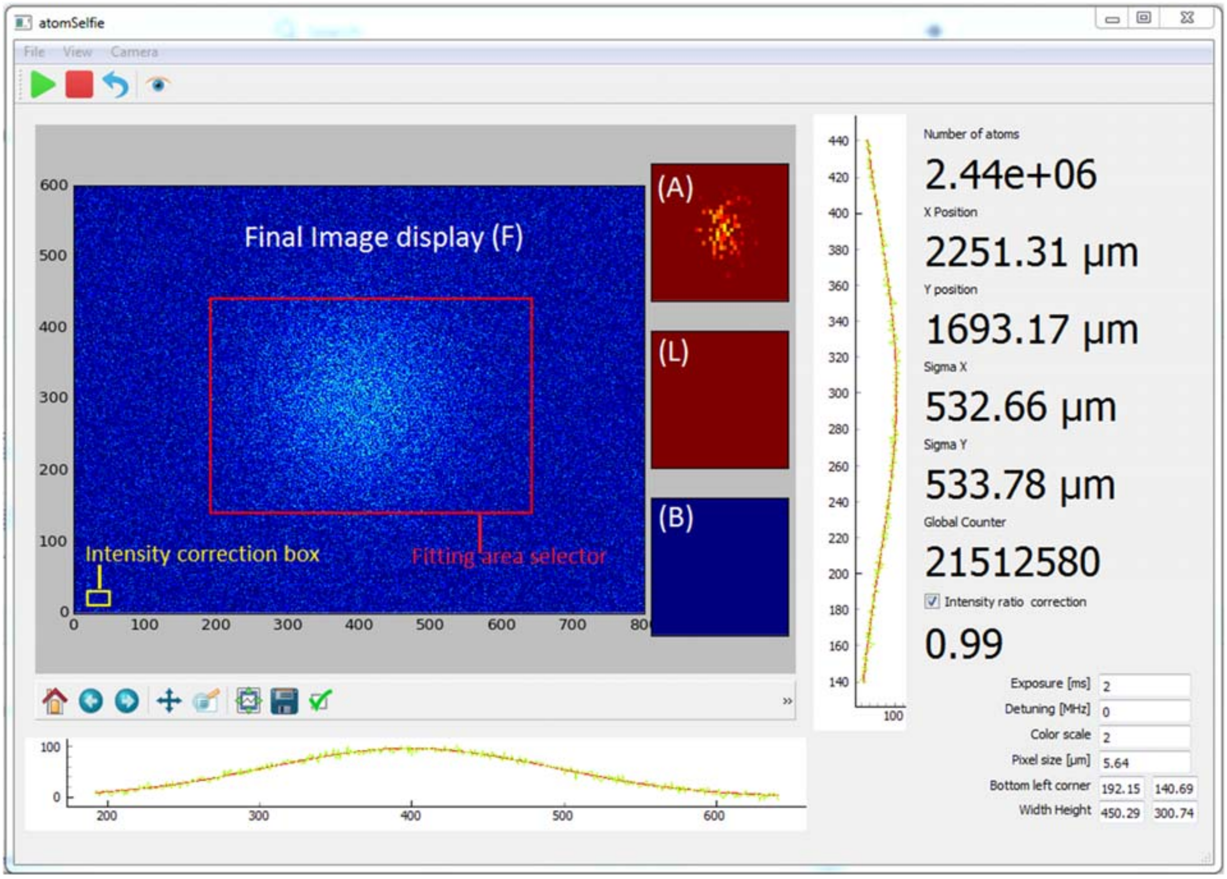


Fig 2: The Matplotlib interactive navigation toolbar.

We also introduced an intensity ratio correction in the final image which selects a small region near the boundary of A and L and compares the intensities. Ideally the ratio should be 1. But in a real scenario it fluctuates and we multiply F by this ratio to get a corrected image. The intensity ratio correction functionality can be turned on(off) by checking(unchecking) the 'Intensity ratio correction'-checkbox. When it is checked, the region of the images where the intensities are compared is displayed as a yellow rectangle. This region can be changed from the configuration file.

The final image matrix is summed along the x and y axes and a Gaussian function is fitted using the numpy.curve_fit module. The summed data and the fitted functions are displayed around the Matplotlib container. Previously when we were using the lmfit module, the program was crashing after some iterations. It was fixed by changing the fitting module to numpy.curve_fit (see Fig 3).

We have made the provision of fitting a particular portion of the main image with the fitting function. To do this one needs to drag the mouse in the desired region of the image. A red rectangle will be visible once the button is released with the corners of the rectangle being the starting and ending points of the drag. The fitting is performed within this region. The shape of the rectangular region can also be changed by entering the coordinates of the bottom left corner and the width and height into the textboxes on the bottom right corner of the GUI. It can also be changed from the configuration file. This functionality of putting a rectangle can be turned off and on just by clicking the mouse.



Number of atoms
2.44e+06
X Position
2251.31 µm
Y position
1693.17 µm
Sigma X
532.66 µm
Sigma Y
533.78 µm
Global Counter
21512580
☑ Intensity ratio correction
0.99

Exposure [ms] 2
Detuning [MHz] 0
Color scale 2
Pixel size [µm] 5.64
Bottom left corner 192.15 140.69
Width Height 450.29 300.74

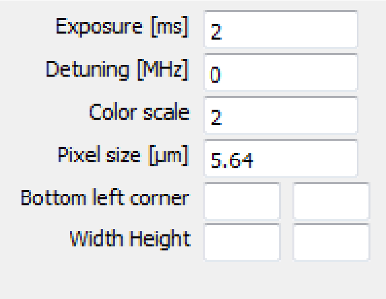
Fig 3: The GUI displaying the images along with the fit and the image parameters.

- **Image parameters:**

1. Display parameters :

All these parameters can be changed by entering the desired values in the textboxes on the bottom right corner of the GUI or by changing in the configuration file. The default values are obtained from the configuration file.

1. Exposure(ms) : This changes the exposure time of the camera shutter.
2. Detuning(MHz): It takes the value of detuning of the imaging light from resonance. This information is used to calculate certain other parameters of the image.
3. Color scale : This changes the color scale of the displayed image.
4. Pixel size(um) : This is used to calibrate the length in real space with the length on the image.



Exposure [ms] 2

Detuning [MHz] 0

Color scale 2

Pixel size [µm] 5.64

Bottom left corner

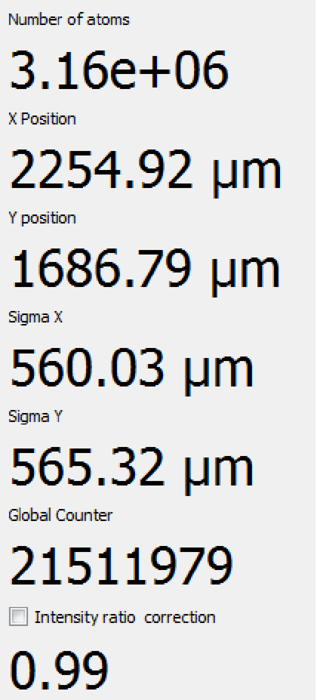
Width Height

Fig 4: The display parameters.

2. Obtained parameters :

The obtained parameters are displayed with bold fonts on the right of the Matplotlib container.

1. Number of atoms(N) : The number of atoms is obtained by summing the final image pixel intensities and dividing by a normalization factor.
2. X-position : The X-position is obtained from the Gaussian fit to the data obtained by summing the image array along the X-axis. It is the mean of the fitted Gaussian.
3. Y-position : The Y-position is obtained from the Gaussian fit to the data obtained by summing the image array along the Y-axis. It is the mean of the fitted Gaussian.
4. Sigma X : Sigma X is the variance of the fitted Gaussian along ths X-axis.
5. Sigma Y : Sigma Y is the variance of the fitted Gaussian along ths Y-axis.
6. Global counter : The global counter is obtained by making a query on the database. It keeps a track of the number of times the experiment was run.



Number of atoms

3.16e+06

X Position

2254.92 µm

Y position

1686.79 µm

Sigma X

560.03 µm

Sigma Y

565.32 µm

Global Counter

21511979

□ Intensity ratio correction

0.99

Fig 5: The obtained parameters.

- **Image saving :**

The images are saved in a folder that is created with the date of the experiment. The path of the folder can be changed from the configuration file. We save the final image as .png file and the image parameters as well as the other three images as a compressed .mat file.

The parameters saved in the .mat file are :

(all the parameters are in SI units)

1. exposureTime : The exposure time of the camera.
2. detuning : Detuning of the imaging light from resonance.
3. pixelSize : Real space length that one pixel corresponds to.
4. sigma0 : The normalization constant for the image.
5. globalcounter : The counter of the run number of the experiment.
6. timestamp : The date and time of the image acquisition.
7. xFitAmplitude : Amplitude of the Gaussian fit of the final image along x-axis.
8. xFitCenter : The center of the Gaussian fit of the final image along x-axis or the position of the cloud.
9. xFitSigma : The variance of the Gaussian fit of the final image along x-axis.
10. yFitAmplitude : Amplitude of the Gaussian fit of the final image along y-axis.
11. yFitCenter : The center of the Gaussian fit of the final image along y-axis or the position of the cloud.
12. yFitSigma : The variance of the Gaussian fit of the final image along y-axis.

Since the image saving process is slower compared to the generation of images, so initially we obtain the files in the current working directory and a separate thread runs to copy the files from the working directory to the desired destination.

An image can also be saved individually using the save functionality of the Matplotlib toolbar.

- **Configuration file :**

We made a separate configuration file (configuration.txt) to make the parameter handling process more convenient. It is a .txt file which can be edited to set different parameters. The default parameters are always the ones set in the configuration file.

The following parameters are set in the configuration file :

```
To set the color scale of the final figure:
vmax_finalfigure=2

To set the mininum and maximum values of the color scale of the images from the camera:
vmin_camerafigures=0
vmax_camerafigures=2**14

To limit the number of iterations while fitting a Gaussian to the image we introduce max_iter. If the
fitting is not good, the number of iterations for fitting might reach a large value leading to the software
to hang:
max_iter=50
```

To set the exposure time of the camera in seconds:

```
exposure_time=2
```

To set the detuning:

```
detuning=0
```

To set the pixelsize:

```
pixelsize=5.6
```

To specify the path of the directory where the images will be stored:

```
filename=r"D:\SVN\RQO\Our Programs\Stefan's camera software\src\dstDir"
```

To set the coordinates of the square for comparing the intensities of L and A (coordinates in integers)

```
left=20
right=30
bottom=10
top=20
```

We display A,L and B as a coarse grained image by skipping some values of the array and to set the coarse graining index:

```
graining_index=20
```

Results

The imaging system was tested by running for 24 hours both with the dummy camera and the real Pixelfly camera with the experiment running.

The test was successful as all the images along with the data were saved and obtained after the experiment.

Improvements

The fitting module sometimes can lead to errors if there is a lot of noise in the image. This can be improved with a better and more sophisticated fitting module. But at the same time one has to also make sure that the program should not become slower in order to use more resources on fitting.

This system works for a particular kind of camera. In general one can make a more robust system that can interface with any kind of camera and hence can be used in a more general setting.

Conclusion

The system performance was satisfactory as it agreed with the data obtained previously from the experiment with an older version of the imaging system. The imaging system could successfully determine the position of the ultracold atom cloud with a precision of a micro meter. It also could calculate the total number of atoms to the correct order of magnitude. This imaging system is expected to reveal new resourceful information about the physics of ultracold atoms in general and Rydberg atoms in particular when it is integrated with a proper experimental setup.

Acknowledgements

This project was done as a part of the DAAD WISE scholarship program 2016. I would like to thank Dr. Sebastian Hofferberth for giving me the opportunity to work with his group. I also thank Asaf Paris Mandoki and Ivan Mirgorodskiy for helping me out with the project.

References

[1] Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming, Mark Summerfield, Prentice Hall, 2007.

[2] PyQt Overviews and Guides, https://wiki.python.org/moin/PyQt/Overviews_and_Guides

[3] Python for Engineers : Your first GUI app with Python and PyQt, http://pythonforengineers.com/your-first-gui-app-with-python-and-pyqt/

Appendix

A. **Hardware specifications:**
Camera : PCO.pixelfly usb ultracompact 14 bit ccd-camera

B. **Software specifications :**
Python : version 2.7.12
Qt : version 4.8.7

C. **Codes :**

- **Atomselfie_deb.py** (the main program) :

```
# conda install pyqt pyqtgraph matplotlib mysql-python scipy
# conda install -c conda-forge lmfit

import sys
import time
from PyQt4 import QtCore, QtGui
from PyQt4.QtCore import pyqtSlot
import matplotlib
import os
from PyQt4.uic import loadUiType
import pyqtgraph as pg
import scipy.io as sio
import numpy as np
import MySQLdb as mysql
from lmfit.models import GaussianModel, ConstantModel
from scipy import stats
from datetime import datetime
from time import strftime
from scipy.optimize import curve_fit
from matplotlib.widgets import  RectangleSelector
import matplotlib.patches as patches
import matplotlib.gridspec as gridspec
from matplotlib.figure import Figure
from matplotlib.backends.backend_qt4agg import (
    FigureCanvasQTAgg as FigureCanvas,
    NavigationToolbar2QT as NavigationToolbar)

#other python file imports
from ImageUpdate import ImageUpdateThread
from FileCopier import FileCopierThread
import configuration as conf

matplotlib.use("Qt4Agg")
Ui_MainWindow, QMainWindow = loadUiType('gui_design_v31deb.ui')
```

```
#==============================================================================
class MainWindow(QtGui.QMainWindow):

  PCO_PICOFLY_USB = 1
  DUMMY_CAM = 2

  # images states
  IM_INIT = 0
  IM_WITH_ATOMS = 1
  IM_WITHOUT_ATOMS = 2
  IM_WITHOUT_LIGHT = 3
  imageState = IM_INIT


  sigma  = u"\u03C3"
  mu = u"\u03BC"
  sigmaX = "%s<SUB>x</SUB>"%(sigma)
  sigmaY = "%s<SUB>y</SUB>"%(sigma)

  sigma0 = 2.906692937721E-13 # m^2, REF: Daniel Steck

  #---------------------------------------------------------------------------
  def __init__(self, parent=None):
    self.isGettingImages = False
    self.isSavingImages = False
    self.saveToFile = False
    self.doFit = False
    self.camIsConnected = False
    self.selectedCamera = 0
    self.camIsSelected = False
    self.enable_numeric_box_entry=True
    self.iratioboxstate=False
    self.start_image_moving()
    self.threads = []

    pg.setConfigOption('background', 'w')  # white plot background
    pg.setConfigOption('foreground', 'k')  # black plot foreground

    QtGui.QWidget.__init__(self, parent)
    self.ui = Ui_MainWindow()
    self.ui.setupUi(self)

    # menu stuff
    self.ui.actionQuit.triggered.connect(self.quitApplication)
  self.ui.actionPixelfly_USB.triggered.connect(self.pcoPicelflyUSBselected)

  # toolbar action buttons
  self.ui.actionStart_capture.triggered.connect(self.startImageAquisition)
  self.ui.actionStop_capture.triggered.connect(self.stopImageAquisition)
  self.ui.actionReset_counter.triggered.connect(self.resetImageCounter)
  self.ui.actionConnect_database.triggered.connect(self.connectToDatabase)
  self.ui.actionDisconnect_database.triggered.connect(self.disconnectFromDatabase)
  self.ui.actionForce_trigger.triggered.connect(self.forceTrigger)

  # matplotlib initialization
  self.center_fig = Figure()
  self.gs=gridspec.GridSpec(3,5)
  self.gs.update(hspace=0.05,wspace=0.05,left=0.05,right=0.99,bottom=0.05,top=0.95)
  self.center_axis=self.center_fig.add_subplot(self.gs[:,:4])
  self.atoms_axis=self.center_fig.add_subplot(self.gs[0,4])
  self.light_axis=self.center_fig.add_subplot(self.gs[1,4])
  self.dark_axis=self.center_fig.add_subplot(self.gs[2,4])
  self.center_canvas = FigureCanvas(self.center_fig )
  self.ui.mplvl.addWidget(self.center_canvas)
  self.toolbar = NavigationToolbar(self.center_canvas, self.ui.mplwindow, coordinates=True)
  self.ui.mplvl.addWidget(self.toolbar)
  #self.center_axis = self.center_fig.gca()


  self.axes_image = self.center_axis.imshow(np.zeros((50,50)), vmin=0,
vmax=conf.vmax_finalfigure, aspect=1, origin='lower' ,interpolation='none') #We initialize the
image with a zero array.
  self.axes_image_atom = self.atoms_axis.imshow(np.zeros((50,50)),
vmin=conf.vmin_camerafigures, vmax=conf.vmax_camerafigures, aspect=1, origin='lower'
,interpolation='none' )
  self.axes_image_atom.axes.get_xaxis().set_ticks([])
  self.axes_image_atom.axes.get_yaxis().set_ticks([])
  self.axes_image_light = self.light_axis.imshow(np.zeros((50,50)),
vmin=conf.vmin_camerafigures, vmax=conf.vmax_camerafigures, aspect=1, origin='lower'
,interpolation='none' )
  self.axes_image_light.axes.get_xaxis().set_ticks([])
  self.axes_image_light.axes.get_yaxis().set_ticks([])
  self.axes_image_dark = self.dark_axis.imshow(np.zeros((50,50)),
vmin=conf.vmin_camerafigures, vmax=conf.vmax_camerafigures, aspect=1, origin='lower'
,interpolation='none' )
  self.axes_image_dark.axes.get_xaxis().set_ticks([])
  self.axes_image_dark.axes.get_yaxis().set_ticks([])

  #get info from the camera
  self.center_canvas.mpl_connect('key_press_event', RectangleSelector(self.center_axis,
self.onselect, drawtype='box'))
  self.click_counter = 0
  self.tlc_x=0
  self.tlc_y=0
  self.brc_x=0
  self.brc_y=0
  self.width=0
  self.height=0
  self.rec=patches.Rectangle((self.tlc_x, self.tlc_y),abs(self.tlc_x-self.brc_x),abs(self.tlc_y-
self.brc_y),fill=False, edgecolor="red", linewidth=2 )
  self.center_axis.add_patch(self.rec)
  self.reciratio=patches.Rectangle((conf.left,conf.bottom),conf.right,conf.top,fill=False,
edgecolor="yellow", linewidth=2 )

  self.ui.txtExposureTime.setText(str(conf.exposure_time))
  self.ui.txtDetuning.setText(str(conf.detuning))
  self.ui.txtPixelSize.setText(str(conf.pixelsize))
  self.ui.txtcolorscale.setText(str(abs(conf.vmax_finalfigure)))

  # line edits
  self.ui.txtExposureTime.textChanged.connect(self.setExposureTime)
  self.ui.txtDetuning.textChanged.connect(self.setDetuning)
  self.ui.txtPixelSize.textChanged.connect(self.setPixelSize)
  self.ui.txtcolorscale.textChanged.connect(self.setcolor_scale)
  self.ui.tlcx.textChanged.connect(self.rectangleChanged)
  self.ui.tlcy.textChanged.connect(self.rectangleChanged)
  self.ui.width.textChanged.connect(self.rectangleChanged)
  self.ui.height.textChanged.connect(self.rectangleChanged)
  self.ui.Iratiocheckbox.stateChanged.connect(self.iratiobox)

  self.initGUIelements()
  #self.selectedCamera = self.PCO_PICOFLY_USB
  self.selectedCamera = self.DUMMY_CAM
  self.connectToCamera()
  self.connectToDatabase()
  #self.globalCounter = 38390237

  self.counter = 0


 #--------------------------------------------------------------------------
 def __del__(self):
  self.disconnectFromDatabase()
  try:
   del self.cam
  except NameError:
   pass
 #______________________________________________________________________________________
 def onselect(self, eclick, erelease):
  self.click_counter=self.click_counter+1

  if self.click_counter%2==1:
   self.tlc_x=eclick.xdata
   #print self.tlc_x
   self.tlc_y=eclick.ydata
   #print self.tlc_y
   self.brc_x=erelease.xdata
   #print self.brc_x
   self.brc_y=erelease.ydata
   #print self.brc_y
   self.rec=patches.Rectangle((self.tlc_x, self.tlc_y),abs(self.tlc_x-self.brc_x),abs(self.tlc_y-
self.brc_y),fill=False, edgecolor="red", linewidth=2 )
   self.center_axis.add_patch(self.rec)

   self.enable_numeric_box_entry=False
   self.ui.tlcx.setText("%.2f"%(self.tlc_x))
   self.ui.tlcy.setText("%.2f"%(self.tlc_y))
   self.ui.height.setText("%.2f"%(abs(self.tlc_y-self.brc_y)))
   self.height=abs(self.tlc_y-self.brc_y)
   self.ui.width.setText("%.2f"%(abs(self.tlc_x-self.brc_x)))
   self.width=abs(self.tlc_x-self.brc_x)
   self.enable_numeric_box_entry=True

   #print(' startposition : (%f, %f)' % (eclick.xdata, eclick.ydata))
   #print(' endposition   : (%f, %f)' % (erelease.xdata, erelease.ydata))

  else:
   self.rec.remove()
 #-----------------------------------------------------------------------
 def initGUIelements(self):
  self.ui.lblAtomNumber.setText("N=0")
  self.ui.lblXPosition.setText("x=0")
  self.ui.lblYPosition.setText("y=0")
  self.ui.lblSigmaX.setText("%s=1"%(self.sigmaX))
  self.ui.lblSigmaY.setText("%s=1"%(self.sigmaY))
  self.ui.lblGlobalCounter.setText("0")

  #self.exposureTime = float(str(self.ui.txtExposureTime.text()))*1.0E-3
  self.exposureTime=(conf.exposure_time)*1.0E-3
  #self.detuning = float(str(self.ui.txtDetuning.text()))*1.0E6
  self.detuning=(conf.detuning)*1.0E6
  #self.pixelSize = float(str(self.ui.txtPixelSize.text()))*1.0E-6
  self.pixelSize=(conf.pixelsize)*1.0E-6
  #self.color_scale = float(str(self.ui.txtcolorscale.text()))*1.0
  self.color_scale=abs(float(conf.vmax_finalfigure))

  self.updateFitX([[0.0],[0.0],[0.0]])
  self.updateFitY([[0.0],[0.0],[0.0]])

  #TODO: remove
  #self.updateImage()

#--------------------------------------------------------------------------
def connectToCamera(self):
  if self.camIsConnected:
    del self.cam
    self.camIsConnected = False

  if self.selectedCamera == self.PCO_PICOFLY_USB:
    import pcoPixelflyUSB as pcoPixelfly
    self.cam = pcoPixelfly.PcoPixelflyUSB()
    #print self.cam

    self.cam.prepareImageCapture()
    self.camIsConnected = True
    self.camIsSelected = True

  if self.selectedCamera == self.DUMMY_CAM:
    import dummyCam
    self.cam = dummyCam.DummyCam()
    #print self.cam

    self.cam.prepareImageCapture()
    self.camIsConnected = True
    self.camIsSelected = True

#--------------------------------------------------------------------------
def startImageAquisition(self):
  self.isGettingImages = True
  #print "updating thread"
  imageAquisition = ImageUpdateThread(self.cam)
  imageAquisition.bufferReady.connect(self.gotNewImage)
  self.threads.append(imageAquisition)
  #print "updated thread"
  imageAquisition.start()

 #-----------------------------------------------------------------------
 def start_image_moving(self):
  #if self.isGettingImages:
  path=conf.filename
  datestr=strftime('%Y\\%m\\%d')
  path=path+'\\'+datestr+'\\Images\\'
  #print path
  try:
   os.makedirs(path)
  except OSError:
   if not os.path.isdir(path):
    raise
  self.imagesaver=FileCopierThread('.',path,self)
  self.imagesaver.start()

#____________________________________________________________________________________

 def gotNewImage(self, buf):
  #print "Getting new image"
  im = self.cam.getImage(buf).T
  #print "Got new image"
  ny, nx = im.shape
  trycounter=0

  #print self.counter, self.counter%3
  self.counter += 1

  for t in self.threads:
   del t
  #print "atomselfie imageState:",self.imageState

  if self.imageState == self.IM_INIT or self.imageState == self.IM_WITHOUT_LIGHT:
   #print "updating image array"
   self.imAtoms = im.copy()
   self.axes_image_atom.set_data(im[::conf.graining_index,::conf.graining_index])
   # self.axes_image_atom.set_extent([0,nx,0,ny])
   self.imageState = self.IM_WITH_ATOMS
   #print "update dimage array"

   trying=True
   while trying:
    trycounter+=1
   #We update the global counter after capturing the first image to avoid race conditions
```

```
    try:
     #print "connecting to detabase"
     dbc = self.db.cursor()
     #print "connected to database"
     dbc.execute("SELECT globalCounter FROM cycledata ORDER BY globalCounter DESC LIMIT
1")
     result = dbc.fetchall()
     #print result
     #self.ui.lblGlobalCounter.setText("%s"%str(result))
     #print "Getting global counter"
     self.globalCounter = int(result[0][0])
     #print "Got global counter"
     dbc.close()
     trying=False

    except Exception as e:
     print "could not fetch globalCounter."+str(e)
     trying=True
     try :
      self.disconnectFromDatabase()
     except :
      pass
     try:
      self.connectToDatabase()
     except:
      pass
    if trycounter>10:
     raise Exception('cannot connect to database after 10 attempts')
     break

  elif self.imageState == self.IM_WITH_ATOMS:
   self.imLight = im.copy()
   self.axes_image_light.set_data(im[::conf.graining_index,::conf.graining_index])
   # self.axes_image_light.set_extent([0,nx,0,ny])
   self.imageState = self.IM_WITHOUT_ATOMS

  elif self.imageState == self.IM_WITHOUT_ATOMS:
   self.imDark = im.copy()
   self.axes_image_dark.set_data(im[::conf.graining_index,::conf.graining_index])
   # self.axes_image_dark.set_extent([0,nx,0,ny])
   self.imageState = self.IM_WITHOUT_LIGHT
   self.updateImage()
  #print "updating canvas"
  self.center_fig.canvas.draw()
  #print "canvas updated"
```

```
    if self.isGettingImages:
      self.startImageAquisition()

  def fitfunction(self,x,amp,sigma,mu):
    return amp*np.exp(-(x-mu)**2/(2*sigma**2))

  def fitxGaussianProfile(self, xData, yData):
    ampfitx=0
    sigmafitx=0
    mufitx=0

    yTot = yData.sum()
    mu0 = (xData*yData).sum()/yTot
    sigma0 = np.sqrt(np.abs(((xData-mu0)**2*yData).sum()/yTot))
    amp0 = yTot*(xData[1]-xData[0])/(np.sqrt(2*np.pi)*sigma0)
    p0 = [amp0,sigma0,mu0]
    try:
      popt,pcov = curve_fit(self.fitfunction,xData,yData,p0,maxfev=conf.max_iter)
      [ampfitx,sigmafitx,mufitx]=popt
    except RuntimeError:
      print "Runtime error. Could not fit data! ",self.globalCounter
      #[ampfitx,sigmafitx,mufitx]=p0
      [ampfitx,sigmafitx,mufitx]=[0,0,0]


    ##try:
    # param = gauss.guess(yData, x=xData)
    #param.update(const.guess(yData, x=xData))
    # model = gauss #for adding const use: model=gauss+const, also uncomment param.update()
    ##model=quad
    # fit = model.fit(yData, param, x=(xData),iter_cb=self.iter_cb)


    ##print fit
    #return fit
    #except :
     # pass
  #------------------------------------------------------------------------


#_______________________________________________________________________________
___
  def updateFitX(self, data):
    xData = data[0]
  yData = data[1]
  raw = data[2]

  self.ui.graphFitX.clear()
  self.ui.graphFitX.plot(x=xData, y=yData, pen=0)
  self.ui.graphFitX.plot(x=xData, y=raw, pen=2)
  self.ui.graphFitX.autoRange()

#---------------------------------------------------------------------------
def updateFitY(self, data):
  xData = data[0]
  yData = data[1]
  raw = data[2]

  self.ui.graphFitY.clear()
  self.ui.graphFitY.plot(x=yData, y=xData, pen=0)
  self.ui.graphFitY.plot(x=raw, y=xData, pen=2)
  self.ui.graphFitY.autoRange()

#---------------------------------------------------------------------------
def updateImage(self):
  # keep this
  self.lightsum=np.sum(self.imLight[conf.left:conf.right,conf.bottom:conf.top])
  self.atomsum=np.sum(self.imAtoms[conf.left:conf.right,conf.bottom:conf.top])
  #self.imLight[:]=[x*atomsum/lightsum for x in self.imLight]

  if self.iratioboxstate==True:
    self.ratio=float(self.atomsum)/self.lightsum
  else:
    self.ratio=1

  self.imLight=self.imLight*self.ratio

  #print atomsum
  #print self.imLight[10:20,30:60]

  num = self.imLight - self.imDark
  denom = self.imAtoms - self.imDark
  frac = num.astype(np.float)/denom.astype(np.float)
  # denom can have NULL values. Division will result in INFINITY
  # so: replace INF with 1.0 before logarithm
  frac[frac == np.inf] = 1.0
  frac[frac == -np.inf] = 1.0
  #print frac
  im = np.log(frac)
  #print im
  im[im == np.inf] = 0.0
  im[im == -np.inf] = 0.0
  #print im.shape
  self.axes_image.set_data(im)
  ny, nx = im.shape
  self.axes_image.set_extent([0,nx,0,ny])
  self.axes_image.set_clim(0.0,self.color_scale)
  self.center_fig.canvas.draw()
  #print [int(self.tlc_x), int(self.tlc_x+self.width),int(self.tlc_y-self.height), int(self.tlc_y)]



  if self.click_counter%2==1:
   im_rec=im[int(min((self.tlc_y+self.height),(self.tlc_y))) :
int(max((self.tlc_y+self.height),(self.tlc_y))),int(min((self.tlc_x),(self.tlc_x+self.width))) :
int(max((self.tlc_x),(self.tlc_x+self.width))) ]

   xSpace = np.linspace(0.0+int(self.tlc_x), int((im_rec.shape[1]-1+self.tlc_x)), im_rec.shape[1])
   ySpace = np.linspace(0.0+int(self.tlc_y), int((im_rec.shape[0]-1+self.tlc_y)), im_rec.shape[0])

  else:
   im_rec=im
   xSpace = np.linspace(0.0, (im_rec.shape[1]-1), im_rec.shape[1])
   ySpace = np.linspace(0.0, (im_rec.shape[0]-1), im_rec.shape[0])

  sumX = np.sum(im_rec, axis=0)   # sum up along rows
  sumY = np.sum(im_rec, axis=1)   # sum up along cols

  fitX = self.fitxGaussianProfile(xSpace*self.pixelSize, sumX)
  fitY = self.fitxGaussianProfile(ySpace*self.pixelSize, sumY)

  dX = [xSpace, fitX[0], sumX]
  dY = [ySpace, fitY[0], sumY]
  #print im_rec
  #print xSpace.shape, ySpace.shape
  #print im
  N = np.sum(im_rec)*(self.pixelSize)**2/self.sigma0

  ####update display information on GUI
  self.ui.lblAtomNumber.setText("%.2e"%(N))
  self.ui.lblXPosition.setText("%.2f %sm"%(fitX[1][2]*1e6, self.mu))
  self.ui.lblYPosition.setText("%.2f %sm"%(fitY[1][2]*1e6, self.mu))
  self.ui.lblSigmaX.setText("%.2f %sm"%(fitX[1][1]*1e6, self.mu))
  self.ui.lblSigmaY.setText("%.2f %sm"%(fitY[1][1]*1e6, self.mu))
  ####for older version of display use this kind of code: self.ui.lblSigmaY.setText("%s=%.2f
%sm"%(self.sigmaY, fitY.params['gauss_sigma'].value, self.mu))
  self.ui.lblGlobalCounter.setText("%d"%(self.globalCounter))
  self.ui.lblIRatio.setText("%.2f"%(float(self.atomsum)/self.lightsum))
  #### print image and the gaussian fits to GUI

  self.updateFitX(dX)
  self.updateFitY(dY)



  self.isSavingImages = True

  trying = True
  while trying:
   try:
    #print "saving image as .mat"
    ###save data to *.mat and *.png file
    sio.savemat('%s_images.mat'%(self.globalCounter),
          {'images':
           {"imageRaw1" :self.imAtoms,
            "imageRaw2" :self.imLight,
            "imageRaw3" :self.imDark
           },
           'exposureTime':  self.exposureTime,
           'detuning':      self.detuning,
           'pixelSize':     self.pixelSize,
           'sigma0':        self.sigma0,
           'globalCounter': self.globalCounter,
           'timestamp':     '%s'%(datetime.now().isoformat()),
           'xFitAmplitude': fitX[1][0]/1.0E-6,
           'xFitCenter':    fitX[1][2]/1.0E-6,
           'xFitSigma':     fitX[1][1]/1.0E-6,
           'yFitAmplitude': fitY[1][0]/1.0E-6,
           'yFitCenter':    fitY[1][2]/1.0E-6,
           'yFitSigma':     fitY[1][1]/1.0E-6,}
          ,do_compression=True)
    #print "saved image as .mat"
    #print "saving image as .png"

matplotlib.pyplot.imsave('%s_images.png'%(self.globalCounter),im,vmin=0,vmax=conf.vmax_fin
alfigure)
    #print "saved image as .png"
    trying = False #Success
   except:
      pass

  self.isSavingImages = False

#---------------------------------------------------------------------------
def stopImageAquisition(self):
  self.isGettingImages = False

#---------------------------------------------------------------------------
def resetImageCounter(self):
  self.imageState = self.IM_INIT

#---------------------------------------------------------------------------
def connectToDatabase(self):
  self.db = mysql.connect(host="rqodatabase",
                          user="rqo", passwd="DotA",
                          db="rqodatabase")
  self.db.autocommit(True)


#---------------------------------------------------------------------------
def disconnectFromDatabase(self):
  del self.dbc
  self.db.close()
  del self.db

#---------------------------------------------------------------------------
def browseDirectory(self):
  dir = str(QtGui.QFileDialog.getExistingDirectory(self, "Select Directory"))
  self.ui.txtDirectory.setText(dir)

#---------------------------------------------------------------------------
def forceTrigger(self):
  if self.camIsSelected:
    self.cam.forceTrigger()

#---------------------------------------------------------------------------
def pcoPicelflyUSBselected(self):
  # TODO: later, auto-deselect all other cameras in menu
  # e.g. self.ui.actionSelectOtherCameras.setChecked(False)
  self.selectedCamera = self.PCO_PICOFLY_USB

#---------------------------------------------------------------------------
def recognizeDirectory(self, contents):
  # TODO: check for valid dir name
  self.selectedDirectory = contents

#------------------------------------------------------------------------
def doFitStateChanged(self):
  if self.doFit:
    self.doFit = False
  else:
    self.doFit = True

#------------------------------------------------------------------------
def getInfo(self):
  i1 = self.cam._llGetTriggerMode()[1]
  self.ui.lblInfo.setText(i1)

#------------------------------------------------------------------------
def setExposureTime(self, content):
  try:
    self.exposureTime = float(content)*1.0E-3
  except ValueError:
    pass

#------------------------------------------------------------------------
def setDetuning(self, content):
  try:
    self.detuning = float(content)*1.0E6
  except ValueError:
    pass

#------------------------------------------------------------------------
def setPixelSize(self, content):
  try:
    self.pixelSize = float(content)*1.0E-6
  except ValueError:
    pass

#------------------------------------------------------------------------
def setcolor_scale(self, content):
  try:
    self.color_scale = float(content)*1.0
  except ValueError:
    pass
#__________________________________________________________________________________________
def rectangleChanged(self, content=None):
  try:
    if self.enable_numeric_box_entry:
```

```
    self.height = float(self.ui.height.text())
    self.width = float(self.ui.width.text())
    self.tlc_x = float(self.ui.tlcx.text())
    self.tlc_y = float(self.ui.tlcy.text())

    self.rec.remove()
    self.rec=patches.Rectangle((self.tlc_x, self.tlc_y),self.width,self.height,fill=False,
edgecolor="red", linewidth=2 )
    self.center_axis.add_patch(self.rec)
    self.center_fig.canvas.draw()

  except ValueError:
   pass

 #--------------------------------------------------------------------------------
 def iratiobox(self):
  if self.ui.Iratiocheckbox.isChecked():
   self.iratioboxstate=True
   self.ratio=self.atomsum/self.lightsum
   self.reciratio=patches.Rectangle((conf.left,conf.bottom),conf.right,conf.top,fill=False,
edgecolor="yellow", linewidth=2 )
   self.center_axis.add_patch(self.reciratio)
   self.center_fig.canvas.draw()

  else:
   self.iratioboxstate=False
   self.reciratio.remove()
   self.ratio=1


#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++
 def quitApplication(self):
  self.disconnectFromDatabase()

  try:
   del self.cam
  except NameError:
   pass

  app.quit()

#===============================================================================
if __name__ == "__main__":
 start_time=time.time()
```

```
  app = QtGui.QApplication(sys.argv)
  mainApp = MainWindow()
  mainApp.show()
  sys.exit(app.exec_())


#________________________________________________________________________________
```

- **DummyCam.py** (program of generating sample images)

```
import numpy as np
from time import sleep, time
#import timeit
from datetime import datetime


class DummyCam():

  IM_INIT = 0
  IM_WITH_ATOMS = 1
  IM_WITHOUT_ATOMS = 2
  IM_WITHOUT_LIGHT = 3
  imageState = IM_INIT
  Nx = 800
  Ny = 600

  sigma0 = 2.906692937721E-13
  pixelSize = 1e-6

  #--------------------------------------------------------------------------
  def __init__(self):
    self.X, self.Y = np.meshgrid(np.linspace(-300,300,self.Ny)*self.pixelSize,np.linspace(-
400,400,self.Nx)*self.pixelSize)

    self.i = 1.0

  #--------------------------------------------------------------------------
  def __del__(self):
    pass


  def prepareImageCapture(self):
    pass
```

```
 def waitForImageReady(self):
  if self.imageState == self.IM_INIT or self.imageState == self.IM_WITHOUT_LIGHT:
   sleep(0.1)
   #sleep(0.1)
  elif self.imageState == self.IM_WITH_ATOMS:
   ##sleep(0.1)
   sleep(0.1)
  elif self.imageState == self.IM_WITHOUT_ATOMS:
   ##sleep(0.1)
   sleep(0.1)

 def forceTrigger(self):
  pass

 def getImage(self,buf):

  #print "dummyCamm imageState",self.imageState
  #k=4*np.random.rand(2)-2
  noise = 0.9

  if self.imageState == self.IM_INIT or self.imageState == self.IM_WITHOUT_LIGHT:
   self.imageState = self.IM_WITH_ATOMS
   self.i += 0.01
   sx = 100.0e-6
   sy = 100.0e-6
   k = [0,0]
   # return np.exp(-np.exp(-((self.X-k[0])**2)/(1.0+self.i)-(self.Y-
k[1])**2/(1.0+self.i)))+0.5*np.random.rand(self.Nx,self.Ny)
   return np.exp(-self.sigma0*(1e5/(2*np.pi*sx*sy))*np.exp(-((self.X-k[0])**2)/(2*sx**2)-(self.Y-
k[1])**2/(2*sy**2)))*(1+noise*np.random.rand(self.Nx,self.Ny))*2**14#with noise
   #return np.exp(-self.sigma0*(1e5/(2*np.pi*sx*sy))*np.exp(-((self.X-k[0])**2)/(2*sx**2)-
(self.Y-k[1])**2/(2*sy**2)))*2**14#without noise
   #return 1-np.exp(-((self.X)**2)/(20.0)-((self.Y-self.i)**2)/(20.0))

  elif self.imageState == self.IM_WITH_ATOMS:
   self.imageState = self.IM_WITHOUT_ATOMS
   return np.ones((self.Nx,self.Ny))*(1+noise*np.random.rand(self.Nx,self.Ny))*2**14

  elif self.imageState == self.IM_WITHOUT_ATOMS:
   self.imageState = self.IM_WITHOUT_LIGHT
   return np.zeros((self.Nx,self.Ny))

 def _llGetTriggerMode(self):
  return ("dummy1","dummy2")
```

- **fileCopier.py** (program for copying files to the destination folder) :

```
from PyQt4 import QtCore, QtGui
import shutil
from os.path import abspath
from os.path import sep
from os import listdir
from time import sleep

#============================================================================
class FileCopierThread(QtCore.QThread):

 sigThreadClosed = QtCore.pyqtSignal(object)
 fileSeparatorChar = sep
 fileEndings = ('mat','png')

 #--------------------------------------------------------------------------
 def __init__(self, srcDir, dstDir, atomselfie):
  QtCore.QThread.__init__(self)
  self.srcDirectory = srcDir
  self.dstDirectory = dstDir
  self.stop = False
  self.isCopying = False
  self.atomselfie = atomselfie

 #--------------------------------------------------------------------------
 def __del__(self):
  while self.isCopying:
   sleep(0.01)

  self.wait()

 #--------------------------------------------------------------------------
 def run(self):
  while not self.stop:
   sleep(1)
   #print "Copying images"
   localMatFiles = self.getFilenames(self.srcDirectory, self.fileEndings)
   if self.atomselfie.isSavingImages==False:
    for f in localMatFiles:
     self.isCopying = True
     try :
      newFile = shutil.move(f, self.dstDirectory
                   + self.fileSeparatorChar
                   + f.split(self.fileSeparatorChar)[-1])
```

```
          except :
            pass
          self.isCopying = False

    self.sigThreadClosed.emit(True)
    self.terminate()

  #------------------------------------------------------------------------
  def getFilenames(self, directory, lineEndings):
    names = []
    for f in listdir(directory):
      if f.endswith(lineEndings): #and f!=str(self.atomselfie.globalCounter)+'_images.mat' and
f!=str(self.atomselfie.globalCounter)+'_images.png' and f!=str(self.atomselfie.globalCounter-
1)+'_images.mat' and f!=str(self.atomselfie.globalCounter-1)+'_images.png':
        names.append(abspath(directory + self.fileSeparatorChar + f))
    return names

#___________________________________________________________________________
```

- **ImageUpdate.py** (program for updating the image in a separate thread) :

```
from PyQt4 import QtCore, QtGui

#============================================================================
class ImageUpdateThread(QtCore.QThread):

  bufferReady = QtCore.pyqtSignal(object)

  #------------------------------------------------------------------------
  def __init__(self, cam):
    QtCore.QThread.__init__(self)
    self.cam = cam

  #------------------------------------------------------------------------
  def __del__(self):
    self.wait()

  #------------------------------------------------------------------------
  def run(self):
    buf = self.cam.waitForImageReady()
    self.bufferReady.emit(buf)

    self.terminate()
```