Diana Gaviria

University of Central Florida

University of Colorado at Boulder, Boulder, Colorado 80309
Contact: dianagaviria16@gmail.com

Introduction

This project is part of the Physics REU summer 2009 program at the University of Colorado. The main objective of the project is to design, test, and implement a new imaging system that will be used for the study of an ultracold gas of molecules. Although the experimental set up already includes an imaging system, a new Pixelfly camera will be implemented in the experiment. Its purpose will be to obtain complementary information along a different axis. The new system will be positioned to capture the top portion of the cloud of atoms. Capturing images of the cloud simultaneously from two angles will provide new information about atom behavior during expansion. The new camera will also be used to study the behavior of multiple clouds trapped in the new optical lattice that is being built. It performs almost as well as the current camera, but it is significantly smaller. Size is a major factor considering the lack of space in the current experimental set up.

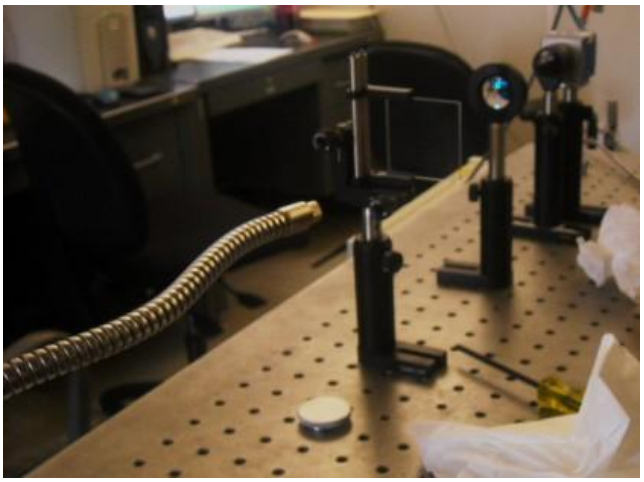
Imaging System

The new imaging system consists of the Pixelfly qe high performance digital 12 bit CCD camera (Figure 1) and a PCI board. The camera has a quantum efficiency of 62% and a 1392 x 1024 pixel resolution, where each pixel size is 6.45 x 6.45 μm^2. The camera's dimensions are 39 mm in width, 39 mm in height and 53 mm in length. Its compact design contributes to its 0.26 kg weight. The camera cost approximately $8000.

Camera/Computer Communication

The first step in accomplishing the project's goals was to establish communication between the camera and the computer. As shown in Figure 2, the initial set up consisted of the camera, a ND4 filter, a PCX focus lens (f= 7.56 cm), a target image, and a white light source. Initially, Visual Basic was tested, but the camera drivers were impossible to access using the LoadLibrary function; one reason could be that

the .ddl files are written in C++, which implements different rules to define pointers. After many trials, it was decided that MATLAB was a more practical computer language to use because it is compatible with the camera drivers. It is also designed for convenient matrix manipulation, which will be more useful for fitting calculations.



The code was written using the latest MATLAB version available (7.8.0). The camera drivers were downloaded from the PCO.imaging corporation website. The main code was divided into six major sections for simplicity (refer to Appendix A for the code). The first part allowed the user to set camera parameters like exposure time, trigger mode, binning, gain, etc. The second part included the camera, memory, and general control functions necessary for collecting an image. The third section called a function that displays the images collected by the previous section. The next three sections called a sequence of functions that perform mathematical operations to obtain optical depth, the region of interest, and Gaussian fits in two dimensions.

To make the code more efficient, the memory control functions were moved around as much as possible to reduce the program running time for taking one image. Using the tic,toc built-in MATLAB function, it was found that the time it takes the program to take one picture is between 327 and 420 ms.

Camera Calibration

Shot Noise

After learning how to control the camera, the next step was to calibrate the camera to find the gain by using the shot noise. The shot noise is a type of electronic noise that occurs when particles like photons or electrons generate detectable random fluctuations in a measurement .[2]The experimental set up was similar to the one shown in Figure 2, except that neither a target image nor a focus lens was used. The light source was pointed directly at the camera through the ND filter. Two main groups of data were collected, one with the hardware gain value set to low, and the other set to high. The data were recorded for a range of exposure times raging from 100 to 10000 μs. To reduce background noise, two images were taken and subtracted for every exposure time recorded. Also, it was noticed that after

being completely covered, the camera still detected some light (dark noise). This dark noise was measured and subtracted in quadrature from every noise value to obtain a more accurate shot noise. The light intensity and shot noise were calculated taking the mean and the standard deviation, with proper error propagation, of the matrix of the resulting image, respectively, as shown in equations (1) and (2).

$$\textbf{Light Intensity}\,[counts] = \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \quad (1)$$

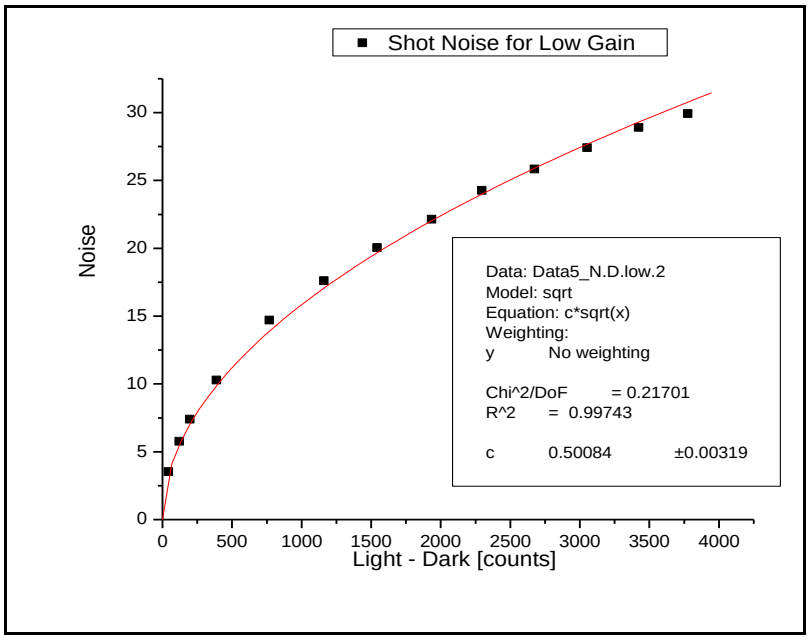
$$\textbf{Noise}\,[counts] = s = \frac{\left(\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2\right)^{\frac{1}{2}}}{\sqrt{2}} \quad (2)$$

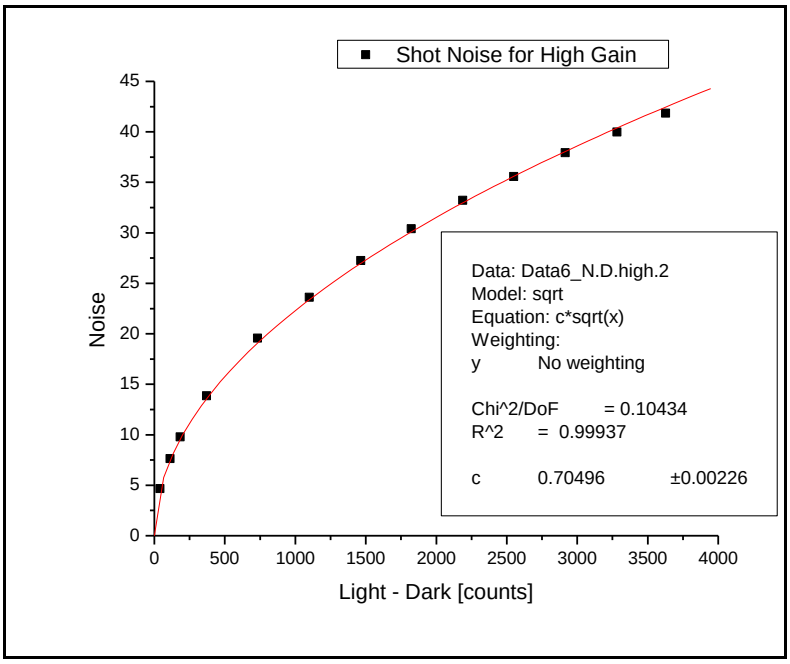
Once these values were plotted, it was confirmed that, in fact, the noise data behaved like shot noise and fit perfectly to a square-root function for both hardware gains. Figures 3 and 4 show this behavior. Because the shot noise scales as the square root of the intensity, the data was fit to equation (3.2), where c represents a constant and L represents the light counts. Using c from the fitting function and equation (4), the camera gain value, or the A/D conversion factor was found to be 4.0 e^-/count for the hardware low gain and 2.04 e^-/count for the high gain.

$$\textbf{Noise} = \sqrt{\frac{Number\ of\ electrons}{gain\left[\frac{e^-}{count}\right]}} \quad (3)$$

$$\textbf{Noise} = c\sqrt{L\,[count]} \quad (3.2)$$

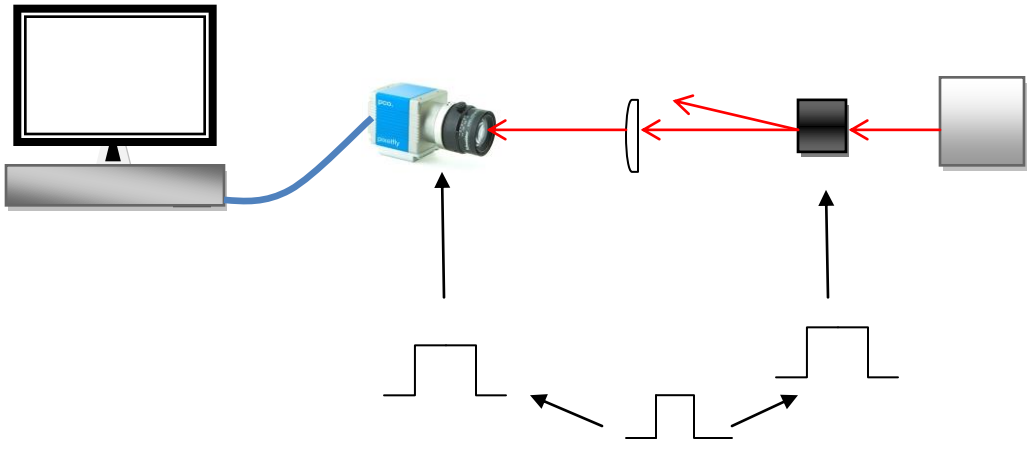
$$\textbf{Gain} = g\left[\frac{e^-}{count}\right] = \frac{1}{c^2} \quad (4)$$



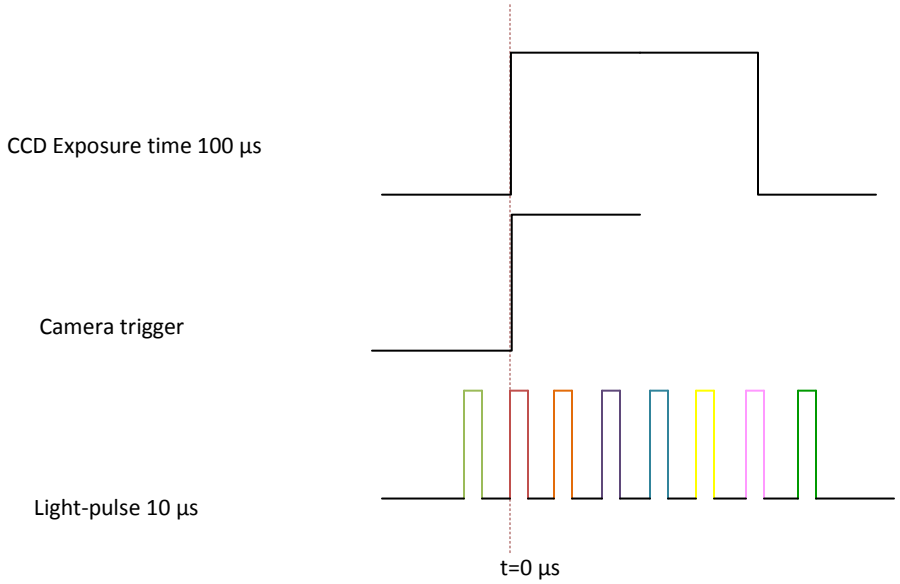


Internal Camera Delay

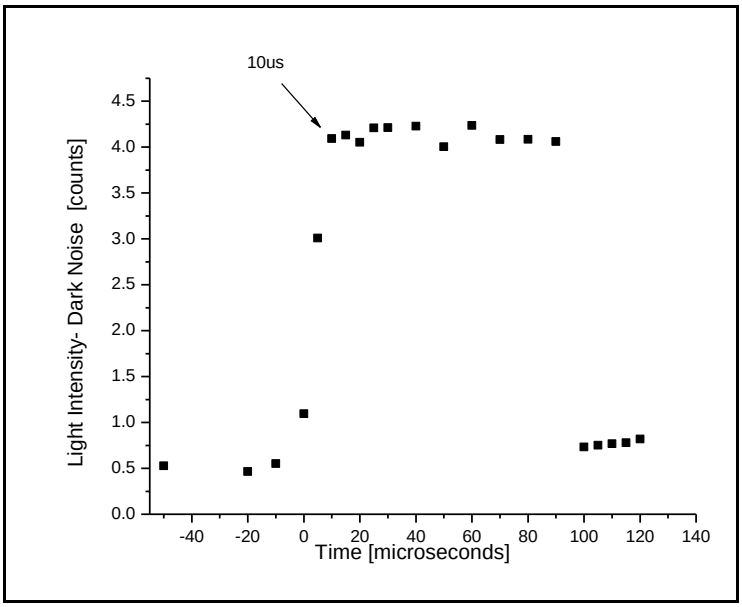
The next step was to find the internal camera delay. This is the time it takes the camera to actually take the picture after it has been triggered. Figure 5 shows a diagram of the set up used to measure this time delay.



The square functions in the set-up diagram represent the three function generators that were utilized; one to trigger the camera, one to trigger the light, and the third one to trigger both of them. The CCD exposure time of the camera was kept at 100 µs while the light pulse was moved at different times to observe the effect on the light intensity read out. Figure 6 demostrates how the pulses were coordinated; each color in the light-pulse square wave represents different times at which measurements were recorded.

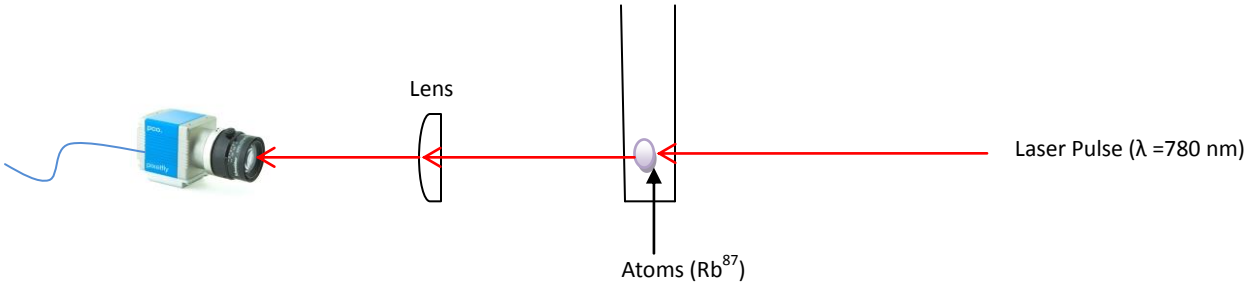


As shown in Figure 7 the light intensity versus time data was plotted. From this graph, it can be concluded that the internal camera delay time is approximately 10 µs.

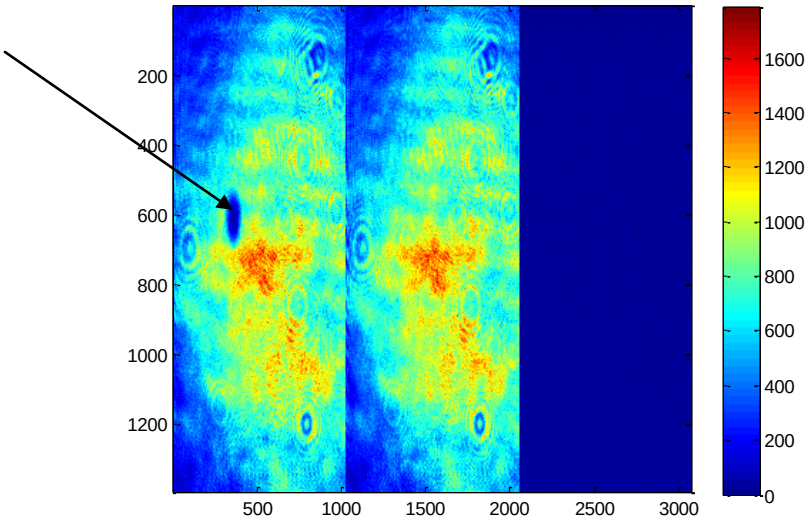


Test Experiment

After the timing and gain calibration the next step was to test the new imaging system in the actual experiment. Figure 8 shows a very basic diagram of the experimental set up.



The test experiment consisted of taking Images of ^{87}Rb clouds for a variety of time-of-flight (TOF), ranging from 7 to 18 ms during the expansion period. For every TOF, three images were taken: shadow, light, and dark. A new cloud was prepared for every expansion time recorded. Figure 9 shows a sample image from the raw data.



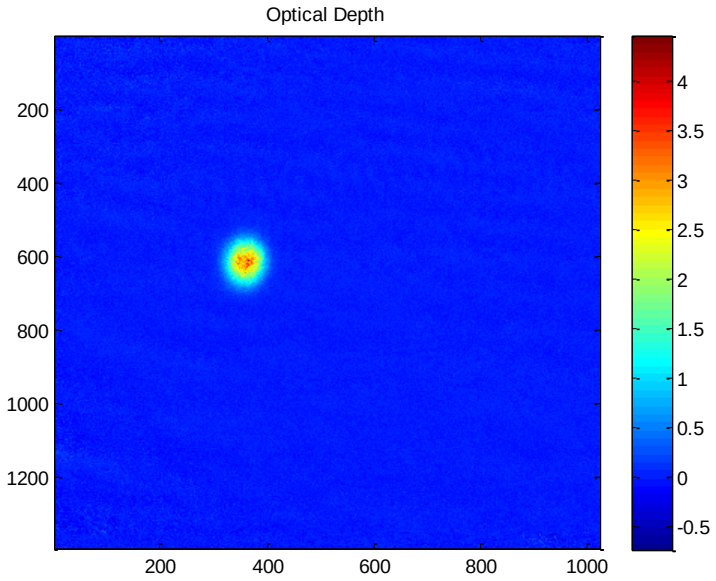
Data Analysis

The last three sections of the main code comprise mathematical operations to analyze and obtain information from the images. Applying Beer's law, the first of these sections computes the optical depth using equation (5.2). Figure 10 shows the resulting OD image of ^{87}Rb atoms at 7 ms TOF.

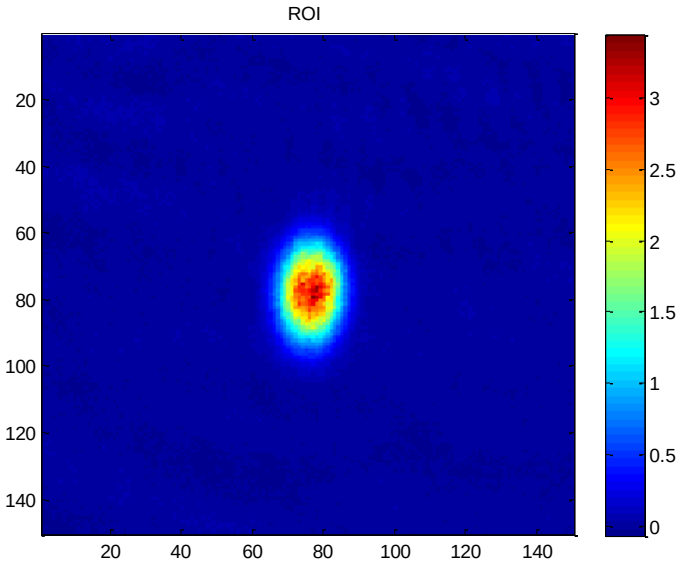
$$\textbf{Beer's Law} = \textbf{Absorbance (gas)} = \ln\left(\frac{I_0}{I}\right),$$

$$\textbf{Optical Depth} = OD = \ln\left(\frac{I_0}{I}\right) = \ln\left(\frac{L-D}{S-D}\right),$$

where I_0 is the intensity of the radiation at the source, and I is the observed intensity after a given path.



The next section of the code is designed to select a region of interest (ROI) using binned data. The OD image previously calculated was binned to increase the signal-to-noise ratio. Figure 11 shows a sample ROI image.



The last section of the code consists of a Gaussian surface-fitting routine. The traces at x = xcenter and y = ycenter are fit to the Gaussian surface function shown below:

$$\textbf{Gauss Surface} = z(x,y) = A * e^{\frac{-(x-xc)^2}{(2*\sigma_x)^2}} * e^{\frac{-(y-yc)^2}{(2*\sigma_y)^2}} + b + mx * x + my * y \ .$$

For each Gaussian fit, there are eight parameters used to describe it: the amplitude (OD peak), x-width, y-width, x-center, y-center, background noise (b), slope in the x direction (mx), and slope in the y direction (my). Initial estimates of these parameters are generated by a function inside the code; they are approximated as follows:

Amplitude or OD peak: is estimated by finding the maximum light intensity [count] minus the average background noise in the matrix of the OD image.

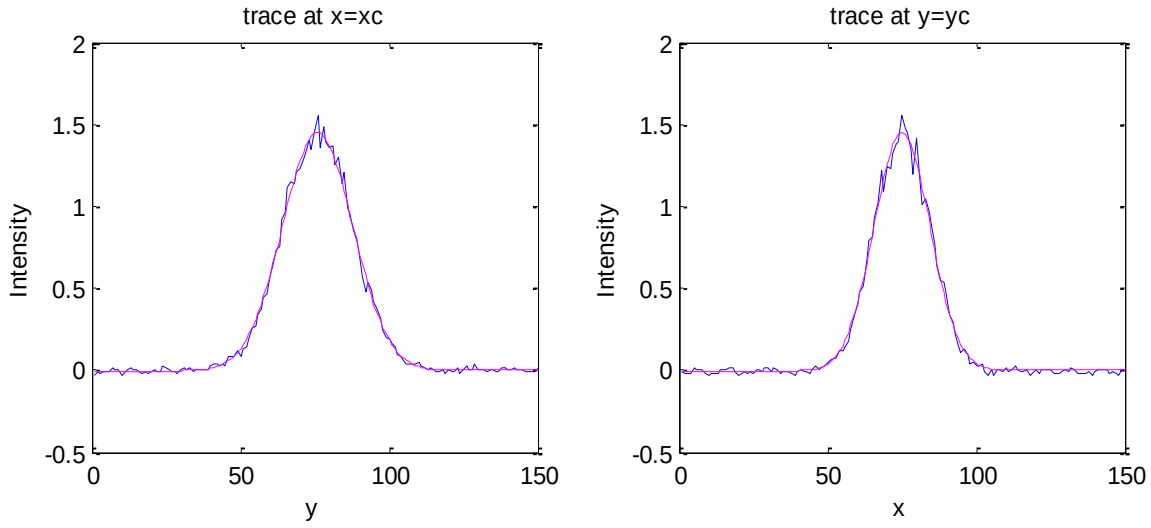
x-center and y-center: are approximated by locating the position coordinates of the OD peak.

Background noise average (b): is estimated by taking the light intensity [count] average of the four small corners of the ROI image.

mx and my: the slopes are expected to be low, so both are approximated to 0.0001.

x-width (σx) and y-width (σy): are estimated using while loops by approximating the distance between the amplitude and the value at which the light intensity is approximately 18% of the amplitude.

When a fit is found, the routine returns these eight parameters. It is critical to have good initial guesses to produce a good fit. Figure 12 shows the Gaussian fits (pink) in both traces of the ^{87}Rb cloud at 12 ms TOF.



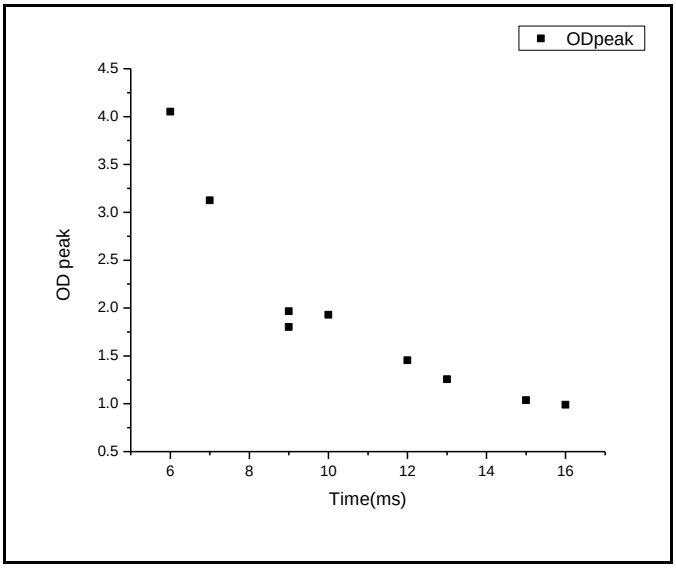
87

Ultracold Gas Information

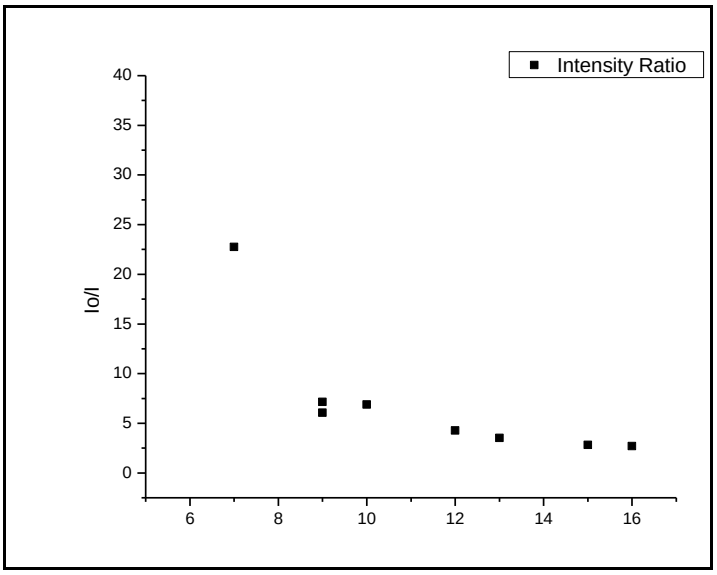
Information about the ^{87}Rb cloud is evaluated by using the Gaussian surface fit, which includes light absorption, the number of atoms, temperature, cloud velocity, and cloud size.

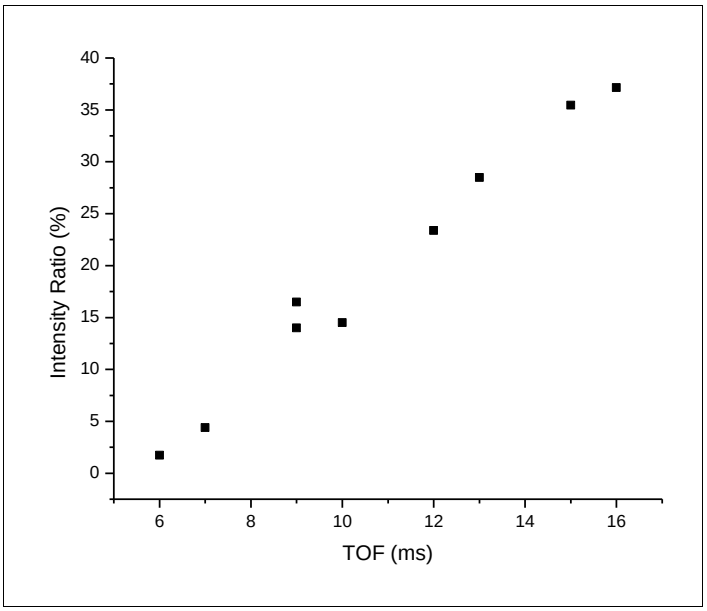
Light Intensity

The light absorption at the center of the cloud can be evaluated by looking at the OD peak. From the plot of the OD peak versus time (Figure 13), it is observed that the OD peak decreases with time. This behavior is expected because as time increases, the cloud density decreases, and the number of atoms remains constant.



In addition, by applying Beer's law and solving equation (5.2) for the intensity ratio, it can be seen that the intensity observed after the beam has passed through the atoms increases with time (Figure 14 and 15). Comparing both TOF extremes, it is estimated that at 6 ms, only 2% of the intensity of the beam is observed; at 16 ms, the intensity observed is 37%. This behavior is expected because as the TOF increases, the cloud becomes less dense; consequently more light can pass through the cloud.





Temperature

The temperature can be measured from the momentum distribution, which is the kinetic energy of the cloud, and the spatial distribution, which is the potential energy of the cloud. Equations (7) and (8) represent the kinetic and potential energy, respectively.

$$\frac{1}{2}k_B * T = \frac{1}{2}m\bar{V}_x^2 ,$$

$$\frac{1}{2}k_B * T = \frac{1}{2}m * \omega^2 \overline{x^2},$$

where T is temperature, m is the mass of an ^{87}Rb atom, ω is the frequency of the trap, and k_B is Boltzmann's constant.

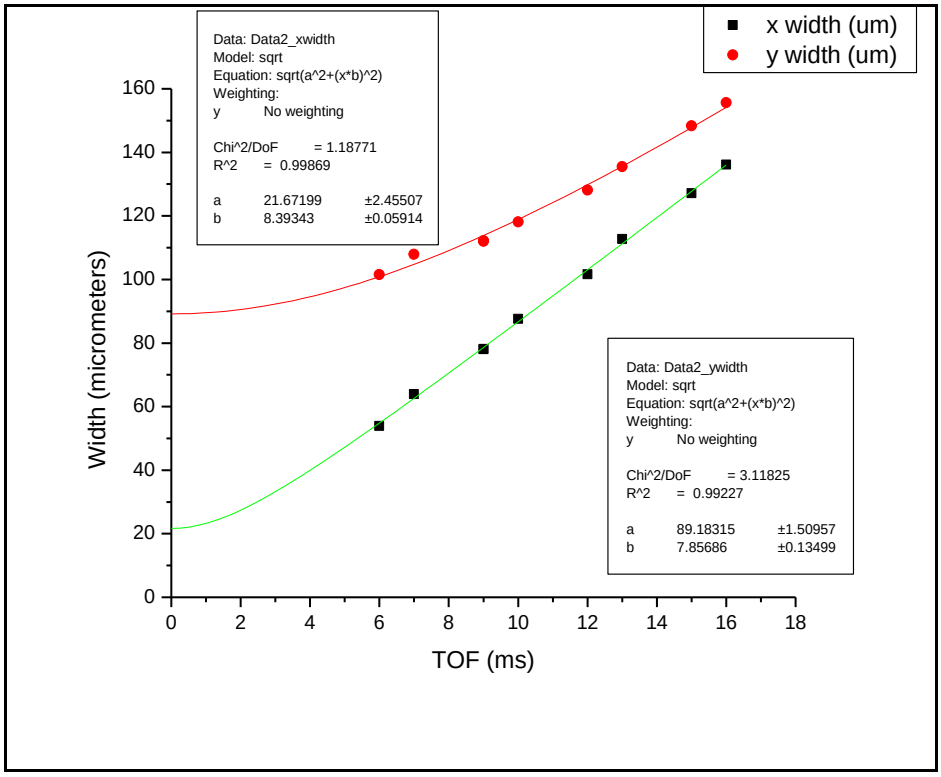
For a Gaussian distribution σ is $\sqrt{\overline{x^2}}$. So,

$\overline{x^2} = \sigma_x^2$, for TOF =0, and

$\overline{v^2} = \frac{\sigma_x^2}{t^2}$, for TOF= ∞.

Then, at any TOF $\sigma_x^2 = \overline{x^2} + \overline{v^2} * t^2$.

Therefore velocity can be obtained from the plot of σ versus time. Figure 16 shows the change in width with time data fit to equation (9) in both dimensions.



From the fitting, the velocities were found to be 8.39(5) mm/s in the x direction and 7.8(1) mm/s in the y direction. Using equation (7), the temperatures in both directions were calculated to be 0.722 μK for x, and 0.638 μK for y. Although the system is in thermoequilibrium, the temperatures obtained are not

identical because of some fitting issues at lower TOF. The data obtained at 6 and 7 ms is suspected to be saturated.

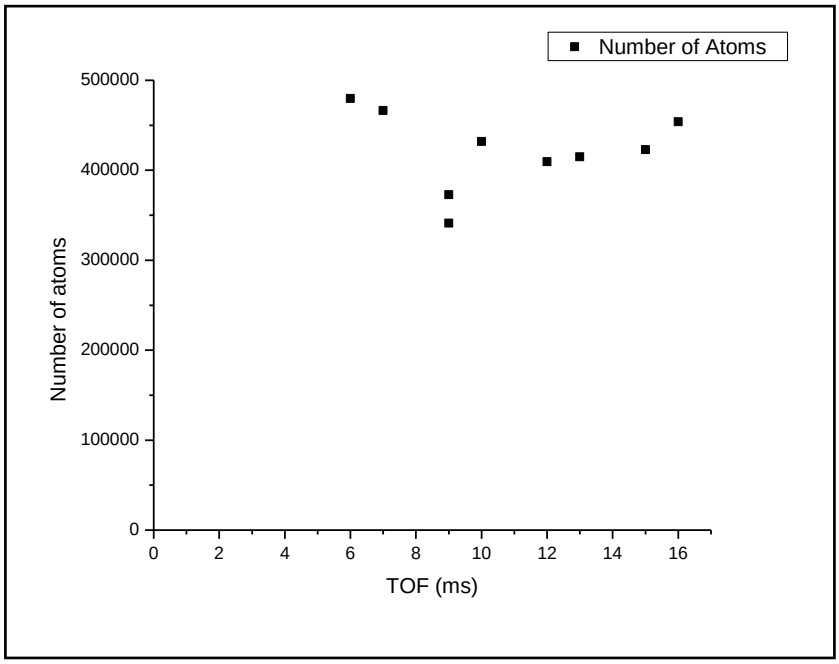
Number of Atoms

The number of atoms in the cloud is calculated from the quantum mechanical properties of the ^{87} Rb atoms using beer's law. This number can be estimated by applying the experiment and fitting parameters into equation (10).

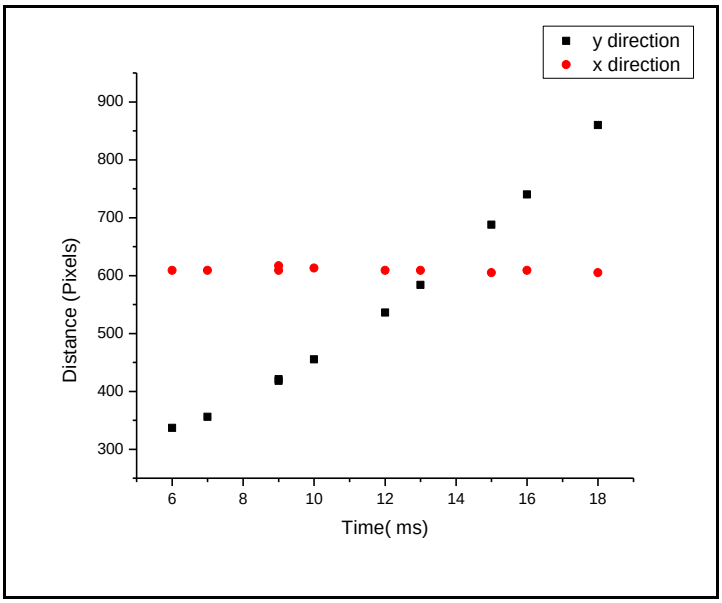
$$\boldsymbol{Number\ of\ Atoms} = \frac{(2\pi)^2 * OD_0(1+\delta^2) realpix^2 * \sigma_x \sigma_y}{3\lambda^2} \quad ,$$

where δ is the detuning in half line widths, realpix is the size of a real pixel, and λ is the wavelength of the beam.

From Figure 17, it can be seen that the data varies only slightly with respect to time because the number of atoms should remain the same even though the density changes. The minor discrepancy between the number of atoms is due to some possible variation between clouds.



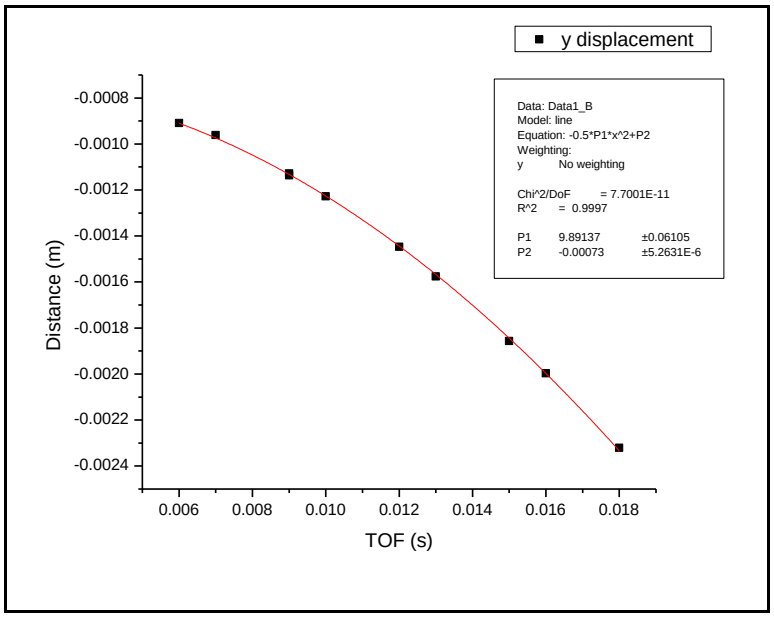
By plotting the position of the center of the cloud versus time, it was observed that the cloud was moving only in the y direction (Figure 18).



Cloud Displacement

Because the change in displacement is a quadratic function of time when acceleration is constant, the y data was fit to the equation of motion shown below to measure the constant acceleration in which the cloud was moving:

$$y(t) = y(0) - \frac{1}{2} * g * t^2 ,$$



From the fitting parameters it is observed that the cloud is moving with a downward acceleration of 9.89(6) m/s^2. In other words, the cloud's acceleration is due to gravity. This result also implies that the magnification used is right.

Conclusion

The new camera system performance is satisfactory. The results obtained from the data agree with previously obtained information from the old imaging system. The code developed works fine for absorption images. In addition, the calibration of the camera gain was successfully accomplished by using the measured camera shot noise. The camera was also tested in the real experimental set up. Data collected the test experiment yield important information about the ^{87}Rb atoms. The images of the cloud disclosed many aspects about the cloud's behavior during the expansion time. The Gaussian surface fit allowed for the calculation of measurements like the temperature and the number of atoms. The limited amount of time did not allow for more, but, in general, the main goals were accomplished and good results were obtained. The new imaging system is expected to reveal innovative information once it is integrated in the experimental set up.

The code routine works fine with the latest MATLAB version, but it still needs improvements to take full advantage of the camera features. Although the double shutter mode was tested with a simple code routine, the current code must be modified to apply this mode. In addition, the selection method of initial guess values for the Gaussian surface fit should be improved because in the current code, a good fit depends on how close these guesses are to the desired solution. Furthermore, a more user friendly software can be developed using MATLAB's Graphical User Interface (GUI).

References

1. PCO.imaging. Accesed 25 July 2009. < http://www.pco.de/sensitive-cameras/pixelfly-qe/>.
2. Paschotta , Rüdiger. "Shot Noise." *Encyclopedia of Laser Physics and Technology*. Web.1 Aug 2009. <http://www.rp-photonics.com/shot_noise.html>.
3. "Beer's law." Encyclopædia Britannica. 2009. Encyclopædia Britannica Online. 02 Aug. 2009 <http://www.britannica.com/EBchecked/topic/58441/Beers-law>.

Appendix A. Code

```
clear all
close all

%Number of times you want to run the experiment
nr_of_exp=1;
file_nr=16;
%Board number is 0 for one camera
board_number=0;
nr_of_images=3;

% Set camera parameters
exptime=1000; %exposure time in microseconds
%mode= 17;   % Single asynchron shutter, software trigger
mode=16;    % Single asynchron shutter, Hardware trigger
explevel=0; % Set level in (%) which time to stop the auto exposure mode,...
            % only valid if auto exposure mode is set
hbin=0;     % Sets horizontal binning and region of camera
vbin=0;     % Sets vertical binning of the camera
gain=1;     % Sets gain value of the camera
bit_pix=12; % Sets how many bits per pixel are transfered
waittime_ms=exptime/1000+1000; %maximum amount of time to wait for the image in ms

for x=1:nr_of_exp

    %Initialize Camera

[board_handle,ret_bufnr,image_size,bufaddress,image_width,image_height]=InitializeCamera(board_nu
mber,nr_of_images,exptime,mode,explevel,hbin,vbin,gain,bit_pix,waittime_ms);

    %Collect Images

[image_stack]=CollectImages(bit_pix,image_width,image_height,nr_of_images,board_handle,ret_bufnr,
image_size,waittime_ms,bufaddress);

    %Close Camera
    [board_handle]=StopCamera(board_handle,ret_bufnr);



    %% Display images
    [nr_of_images]=DisplayImages(nr_of_images,image_stack,x);

    %% Optical Depth
    image=double(image_stack);
    ODimage=log((image(:,:,2)-0.*image(:,:,3))./(image(:,:,1)-0.*image(:,:,3)));
    %to prevent getting infinity values
    indices = find(ODimage==Inf); % help find
    ODimage(indices) = 0;

    figure
    imagesc(ODimage)
    title(['Optical Depth ',num2str(x)])
    colorbar

    %% Select out region of interest (ROI)
    xbin=4;
    ybin=4;
    xc=362;
    yc=609;
    xsize= 160;
```

```
    ysize=200;
    data=ROI(ODimage,xbin, ybin,xc,yc,xsize,ysize);

    figure
    imagesc(data), colorbar
    title(['ROI ',num2str(x)])

      %Saves the stack of pictures
    filename=['data folder/imagestack#' num2str(file_nr) '_' date '.txt'];
    dlmwrite(filename,image_stack);
    msg=['wrote image' filename]

    filename=['data folder/ODimageROI#' num2str(file_nr) '_' date '.txt'];
    dlmwrite(filename,data)
    msg=['wrote ' filename]



    %% fit the optical depth image
    [test2]=Fitting(data);

end




function
[board_handle,ret_bufnr,image_size,bufaddress,image_width,image_height]=InitializeCamera(board_nu
mber,nr_of_images,exptime,mode,explevel,hbin,vbin,gain,bit_pix,waittime_ms)

comment=0;

% Check if library has been already loaded
if not(libisloaded('PCO_PF_SDK'))
 loadlibrary('pccamvb','pccamvb.h','alias','PCO_PF_SDK');
end

% Initialize Camera
[error_code,board_handle] = pfINITBOARD(board_number);
if(error_code~=0)
 disp(['Could not initialize camera. Error is ',int2str(error_code)]);
end

error_code=pfSETMODE(board_handle,mode,explevel,exptime,hbin,vbin,gain,0,bit_pix,0);
if(error_code~=0)
 disp(['SETMODE failed. Error is ',int2str(error_code)]);
 pfCLOSEBOARD(board_handle);
 return;
end
% Get the ccd size
[error_code,ccd_width,ccd_height,image_width,image_height,bit_pix]=pfGETSIZES(board_handle);
if(error_code~=0)
 disp(['GETSIZES failed. Error is ',int2str(error_code)]);
 pfCLOSEBOARD(board_handle);
 return;
end

% Get image size
image_size=image_width*image_height*floor((bit_pix+7)/8);

% Create buffers
bufnr=-1; %-1 is to create a new buffer
[error_code, ret_bufnr] = pfALLOCATE_BUFFER(board_handle, bufnr, image_size);
if(error_code~=0)
 disp(['ALLOCATE_BUFFER failed. Error is ',int2str(error_code)]);
 pfCLOSEBOARD(board_handle);
 return;
end

% Map buffer
```

```
[error_code,bufaddress] = pfMAP_BUFFER_EX(board_handle,ret_bufnr,image_size);
if(error_code~=0)
 disp(['MAP_BUFFER_EX failed. Error is ',int2str(error_code)]);
 pfFREE_BUFFER(board_handle,ret_bufnr);
        pfCLOSEBOARD(board_handle);
 return;
end

% Start Camera
error_code=pfSTART_CAMERA(board_handle);
if(error_code~=0)
 disp(['START_CAMERA failed. Error is ',int2str(error_code)]);
 pfUNMAP_BUFFER(board_handle,ret_bufnr);
 pfFREE_BUFFER(board_handle,ret_bufnr);
 pfCLOSEBOARD(board_handle);
 return;
end
```

```
function
[image_stack]=CollectImages(bit_pix,image_width,image_height,nr_of_images,board_handle,ret_bufnr,
image_size,waittime_ms,bufaddress)
comment=0;

% Allocate Matlab image stack
if(bit_pix==8)
 image_stack=zeros(image_width,image_height,nr_of_images,'uint8');
else
 image_stack=zeros(image_width,image_height,nr_of_images,'uint16');
end

for imanr=1:nr_of_images
    tic
 % Set the buffer int he working list of the driver
 error_code=pfADD_BUFFER_TO_LIST(board_handle,ret_bufnr,image_size,0,0);
 if(error_code~=0)
  disp(['ADD_BUFFER_TO_LIST failed. Error is ',int2str(error_code)]);
  break;
 end

 % Trigger the camera to start the image
 error_code=pfTRIGGER_CAMERA(board_handle);
 if(error_code~=0)
  disp(['TRIGGER_CAMERA failed. Error is ',int2str(error_code)]);
  break;
 end
```



```
Break doub <<26.024 433tus)Errrror code)1);   5    8    .    4    r    r    o    r          c    o    d    e    )
```

```
  end

  if(error_code~=0)
      disp(['WAIT_FOR_BUFFER failed. Error is ',int2str(error_code)]);
  else
      if(comment)
          disp([int2str(imanr),'. image grabbed to buffer ',int2str(ima_bufnr)]);
      else
          disp([int2str(imanr),'. image grabbed ']);
      end
  end
  if(ima_bufnr<0)
      error_code=1;
  end

 if(error_code==0)
  if(comment)
   disp('call pfCOPY_BUFFER, copy the data from the buffer to the Matlab image stack');
  end
  [error_code,image_stack(:,:,imanr)]=
pfCOPY_BUFFER(bufaddress,bit_pix,image_width,image_height,image_stack(:,:,imanr));
 else
  if(comment)
   disp('call pfREMOVE_BUFFER_FROM_LIST, an error occured remove the buffer from the working
list');
  end
  [error_code]=pfREMOVE_BUFFER_FROM_LIST(board_handle,ret_bufnr);
  if(error_code~=0)
   disp(['REMOVE_BUFFER_FROM_LIST failed. Error is ',int2str(error_code)]);
  end
 end
 toc
end




function [board_handle]=StopCamera(board_handle,ret_bufnr)
% Stop the camera
error_code=pfSTOP_CAMERA(board_handle);
if(error_code~=0)
 disp(['STOP_CAMERA failed. Error is ',int2str(error_code)]);
end

% Unmap the mapped buffer before call to FREE_BUFFER
error_code=pfUNMAP_BUFFER(board_handle,ret_bufnr);
if(error_code~=0)
 disp(['UNMAP_BUFFER failed. Error is ',int2str(error_code)]);
end

% Free buffer memory
error_code=pfFREE_BUFFER(board_handle,ret_bufnr);
if(error_code~=0)
 disp(['FREEBUFFER failed. Error is ',int2str(error_code)]);
end

% Close the driver
error_code=pfCLOSEBOARD(board_handle);
if(error_code~=0)
 disp(['CLOSEBOARD failed. Error is ',int2str(error_code)]);
end




function [nr_of_images]=DisplayImages(nr_of_images,image_stack,x)
    figure('Position',[100,100,1000,300])
    for n=1:nr_of_images
        subplot(1,nr_of_images,n)
        imagesc(image_stack(:,:,n));
```

```
colormap('gray')
```

```
subplot(1,2,2)
plot(data(ytrace,:),'b')
hold on
plot(fit(ytrace,:),'m')
hold off
title('trace at y=yc')
xlabel('x')
ylabel('Intensity')
%legend('Data','Fit')

%residuals
figure
imagesc(data-fit),colorbar
title('Residuals')

function [z] = GaussSurf(SurfGaussGuess,xxx,yyy)

A=SurfGaussGuess(1);
xwidth=SurfGaussGuess(2);
ywidth=SurfGaussGuess(3);
xc=SurfGaussGuess(4);
yc=SurfGaussGuess(5);
b=SurfGaussGuess(6);
mx=SurfGaussGuess(7);
my=SurfGaussGuess(8);

z=A.*exp(-((xxx-xc).^2)./(2*xwidth^2)).*exp(-((yyy-yc).^2)./(2*ywidth^2))+b+mx.*xxx+my.*yyy;
```