ACTIVE OPTICAL SYSTEMS

AN014

Hartmann Wavefront Sensor Accuracy versus Signal to Noise Ratio (SNR) and Pixels Across a Sub-Aperture Diameter

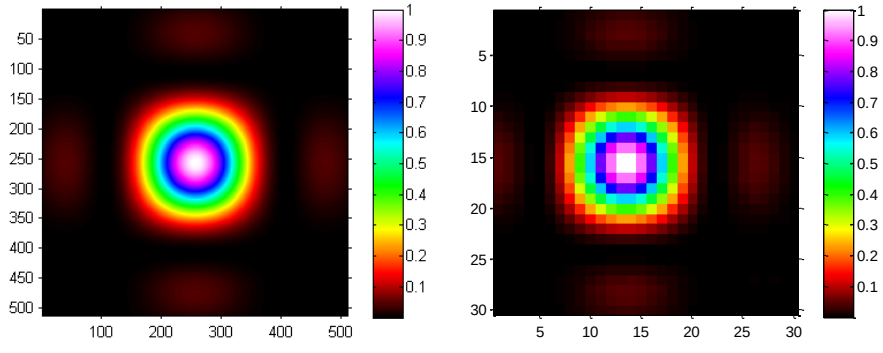
Author: **Justin D. Mansell, Ph.D.**
Active Optical Systems, LLC

Revision: **5/28/10**

In this application note we consider the accuracy of a Hartmann wavefront sensor as it relates to the signal to noise ratio and the number of pixels across a sub-aperture diameter. We have written a simple Matlab simulation to address this question. Because this application note addresses a topic related to the design of wavefront sensors, we have labeled it export controlled.

Simulation Setup

In this simulation we considered a sinc-squared intensity profile that was being moved inside a single sub-aperture. We rendered this intensity profile at a high resolution (512 x 512 samples) and then integrated over the number of samples per pixel to create a sample at the pixel. We did not consider the effect of a reduced fill factor. We setup our high-resolution sampling to have a fixed number of samples per pixel. The image was created to simulate a 150-micron diameter 6.7-mm focal length square aperture lens being illuminated by 633 nm light. The focused spot sinc-squared intensity profile was moved laterally from 0 to 2 pixels in 0.1 pixel increments along the x-axis to simulate varying slope input.



(a) (b)

Figure 1 – (a) High resolution intensity image. (b) Pixelated and shifted image.

The noise was generated using Gaussian spectrum noise with a unity variance. This noise spectrum was multiplied by a varying noise scale factor to create different signal to noise ratios. The code to generate the noise in Matlab is,

```
Inoise = 0.01*Nsf*randn(nx,ny),
```

where Nsf is the noise scale factor that was varied in this simulation from 10 to 0.2. The signal to noise ratio was calculated by taking the ratio of the RMS of the signal (the sinc-squared intensity pattern) to the RMS of the noise. The typical ratio of the peak to valley on the noise and the RMS was 5. When the signals less than 0 were eliminated, the ratio was typically 3. The ratio of the peak to valley to the RMS of the sinc-squared signal was 3.6. Since the measurements were of intensity, which is the field amplitude squared, we simply took the ratio of these RMS values instead of squaring them.

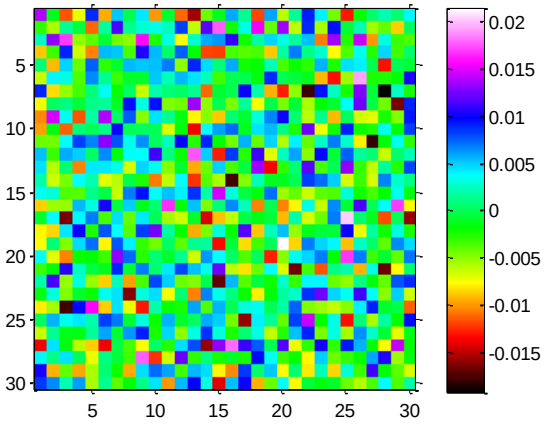


Figure 2 - Sample Noise Pattern

After adding the noise and the signal together, the image was initially normalized by subtracting the minimum and then dividing by the maximum. This procedure created a slight offset, so we moved to zeroing the intensity below zero before dividing by the maximum to normalize.

Then the number of intensity digitization levels (8-bit or 256 levels throughout) was multiplied by the resulting pixilated image and rounded before performing a centroid. In this analysis we only analyzed 8-bit (256 discrete intensity level) images. Every noise level was evaluated at 20 different realizations and the results were averaged to create a representative result.

Position of the spot was determined with a first moment (centroid) with a 5% threshold. For every unique combination of noise amplitude, x-offset, and number of pixels per sub-aperture, the average first moment in the x-axis was subtracted from the real value determined using a pixilated noise-free 5%-threshold first moment. The results for varying x-offset were were squared, averaged, then square-rooted to determine an RMS centroid error.

The RMS slope was calculated as the RMS centroid error divided by the focal length of the lens. The RMS wavefront error (WFE) was determined by multiplying the RMS slopes times the sub-aperture

diameter times a reconstructor factor of 3.0. The Strehl ratio was determined using the Marechal approximation of $\exp(-\phi_{RMS}^2)$.

Results: Scanning the Signal to Noise Ratio

The following graphs summarize the effect of SNR on the centroid accuracy for varying number of pixels across the sub-aperture.

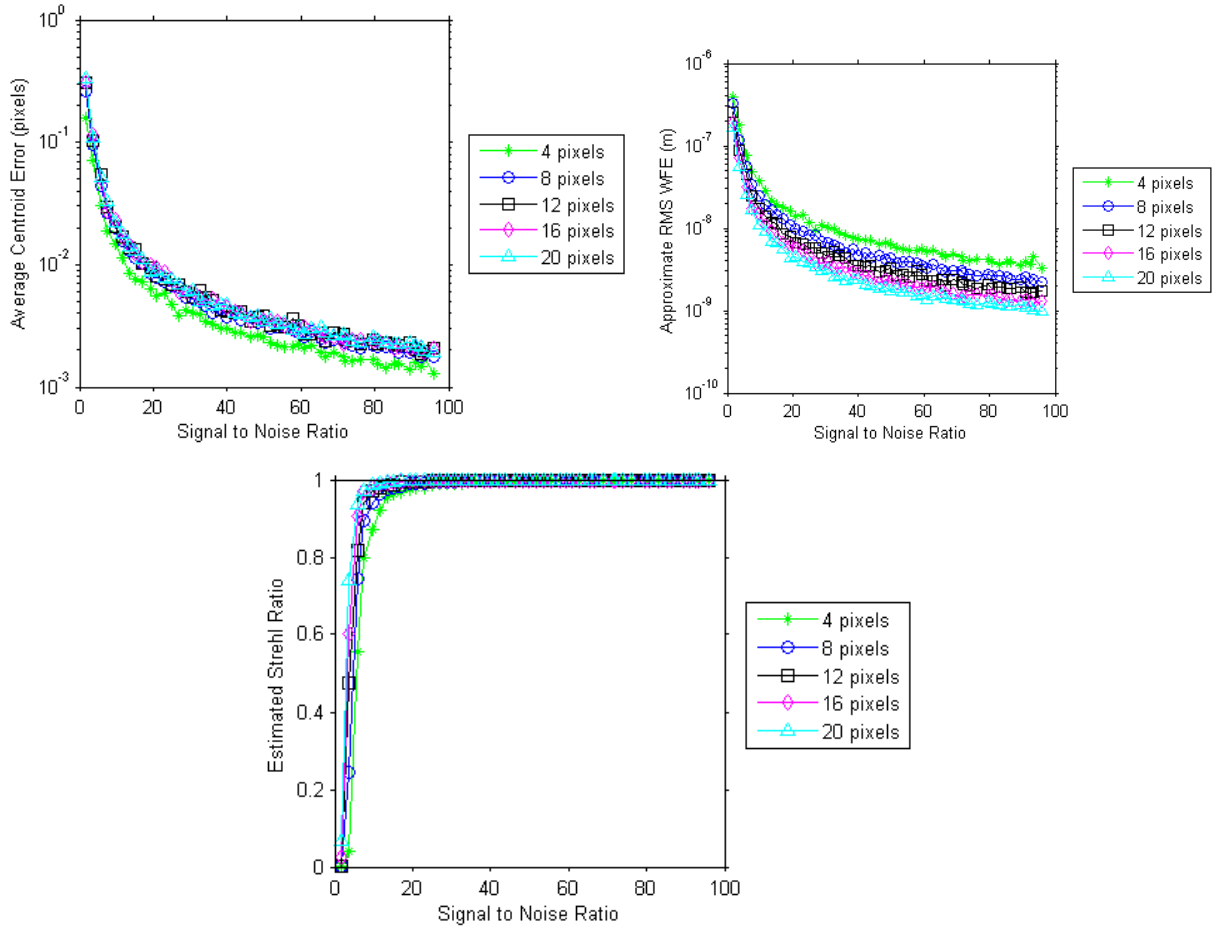


Figure 3 - Results from scanning the signal to noise ratio for a few different number of pixels across the sub-aperture

Results: Scanning the number of Pixels Across the Sub-Aperture

In this set of results, we scanned the number of pixels across a sub-aperture for a few different signal to noise ratios.

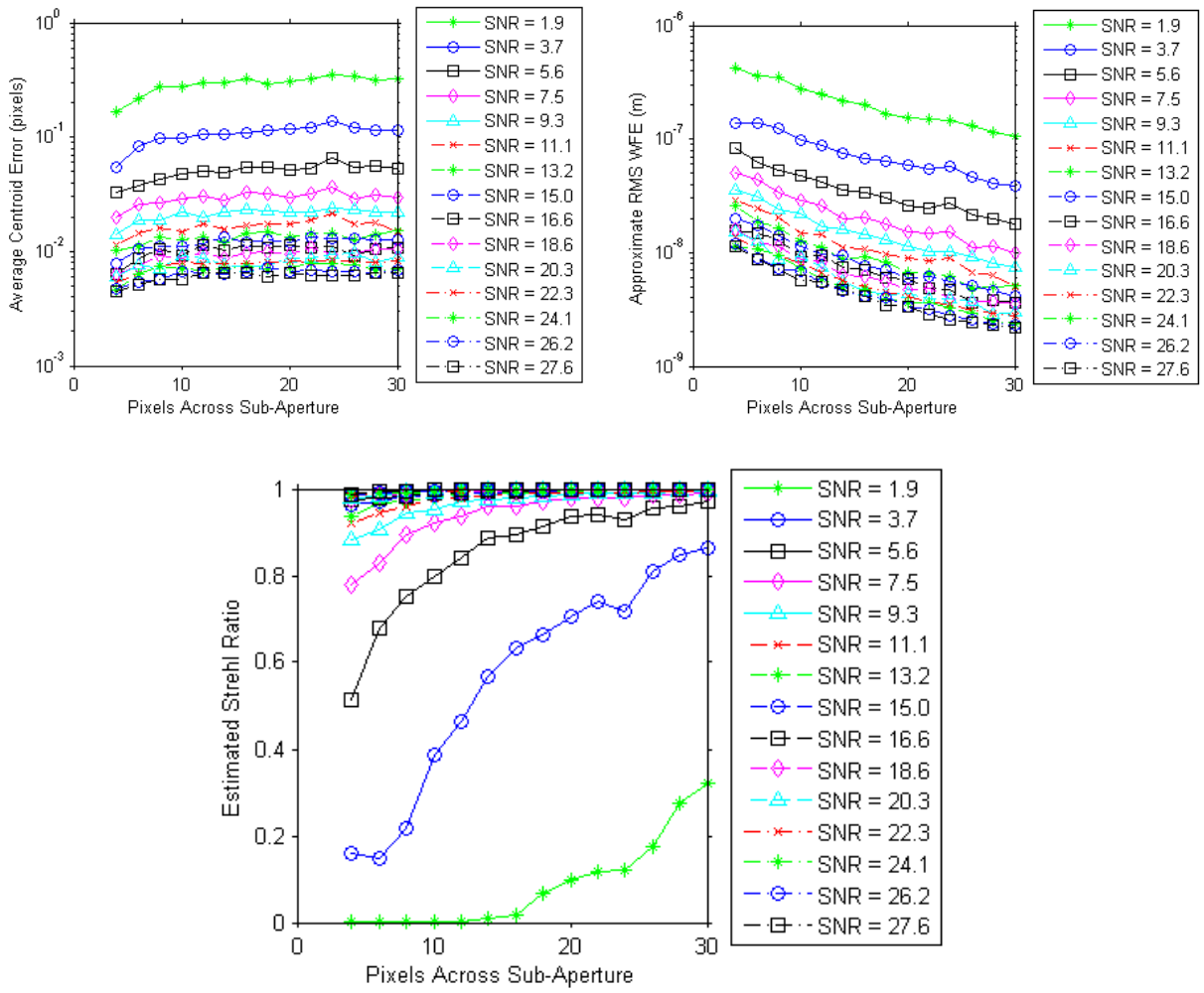


Figure 4 - Results for scanning the number of pixels across a sub-aperture for various signal to noise ratios

Conclusions

The typical rule of thumb is that a centroid is accurate to 1/20 of a pixel. If we consider this to be our goal, it is the case for every pixel case, even down to 4 across a sub-aperture, when the signal to noise ratio is greater than 10. For the case we considered here, this corresponds to a >50% Strehl ratio and RMS wavefront error of less than 50 nm. Therefore, we recommend at least a SNR of 10 for doing Hartmann wavefront sensing. Furthermore, we recommend using at least 4 pixels across a sub-aperture (16 in total).

Appendix A: Matlab Code

```
%routine to study effect of SNR on position accuracy
setup;
ppt=1;

D = 150e-6;
f=6.7e-3;
wavelength=0.633e-6;
bitDepth = 8;
nxy = 512;
threshold=0.05;

type = 2;
stringTitle='detailed scan over pixels';
type = 1;
stringTitle='detailed scan over SNR';

if (type==1)
    %scan over the noise in detail, pixels in low res
    dpixvec = D./[4:4:20];
    NnoiseVec = 1./[0.1:0.1:5.0];
else
    %scan over the pixels and noise in low res
    dpixvec = D./[4:2:30];
    NnoiseVec = 1./[0.1:0.1:1.5];
end;

if (ppt)
    TitleSlidePowerPoint(stringTitle);

    x=whos;
    for ii=1:length(x);
        cmd = sprintf('val=%s;',x(ii).name)
        eval(cmd);
        txt{ii} = sprintf('%s = %.2e',x(ii).name,double(val));
    end;
    TextToPowerPoint(txt,'Variables',12,12);
    clear x;
end

drawing=false;
randn('seed',12345);

levels = (2^bitDepth);
dxy = D/nxy;
g=makegrid(nxy,dxy);
Dff = f*wavelength/D;

I = (sinc(g.xx./Dff*2) .* sinc(g.yy./Dff*2)).^2;
nf; show(I);
```

```
%pixel size
if (drawing)
    nf([346         222        1250         774]);
end;
%profile on;
NpixSize = 0;
%dpixvec = [5.0e-6:5.0e-6:75.0e-6]; %5.0e-6;
for dpix = dpixvec;
    NpixSize=NpixSize+1;
    if (~drawing)
        disp(sprintf('dpix: %e',dpix));
    end;

    dxy = D/nxy;
    %make sure that there are an integer number of samples per pixel
    samplesPerPixel = floor(dpix/dxy)
    dxy = dpix/samplesPerPixel;
    g=makegrid(nxy,dxy);
    Dff = f*wavelength/D;

    %generate pixel grid
    nx = ceil(D/dpix);
    nxvec(NpixSize)=nx;
    ny = ceil(D/dpix);
    [gCAM]=makegridWFS(nx,ny,dpix,dpix);
    xpix = gCAM.xsub;
    ypix = gCAM.ysub;

    %pixelation mask
    IpixelMask = 0 .* I;
    Npix = length(xpix);
    xc0 = nx/2 + 0.5;
    yc0 = ny/2 + 0.5;   %center locations of the spot
    for ii=1:Npix;
        xi1 = xpix(ii) - dpix/2;
        yi1 = ypix(ii) - dpix/2;
        xi2 = xpix(ii) + dpix/2;
        yi2 = ypix(ii) + dpix/2;
        mask = (g.xx>xi1) .* (g.xx<xi2) .* (g.yy>yi1) .* (g.yy<yi2);
        IpixelMask = IpixelMask + mask .* ii;
    end;

    Noffset=0;
    for xoffsetsf = 0:0.2:2.0;
        Noffset=Noffset+1;

        %generate intensity
        I = (sinc((g.xx+xoffsetsf*dpix)./Dff*2) .* sinc(g.yy./Dff*2)).^2;

        %pixelate it
        clear Ipixelated;
        clear Ipix;
        for ii=1:length(xpix)
            Isub = I .* (IpixelMask==ii);
```

```
            Ipix(ii) = sum(Isub(:));
        end;
        Ipixelated = reshape(Ipix,nx,ny);
        Ipixelated = Ipixelated / max(Ipixelated(:));
        [xcIdeal,ycIdeal,x2ideal,y2ideal] =
MomentsWithThreshold(Ipixelated,threshold);

        if (drawing)
            clf;
            subplot(2,2,1);
            imagesc(g.x,g.y,I);
            axis image;
            hold on;
            plot(xpix,ypix,'r*');

            subplot(2,2,2);
            imagesc(Ipixelated);
            axis image;
            colormap(gray);
            title(xoffsetsf);
        end;
        %add noise
        Nnoise=0;

        %NnoiseVec = [0.25:0.25:5.0];
        if (~drawing)
            disp(sprintf('xoffsetsf:%f',xoffsetsf));
        end;
        for Nsf = NnoiseVec;
            Nnoise=Nnoise+1;
            dxavg=0;
            dyavg=0;
            for MC=1:20;
                Inoise = 0.01*Nsf*randn(nx,ny);
                InoiseRMS(MC) = RMS(Inoise);
                Itotal = Ipixelated + Inoise;
                Itotal = Itotal .* (Itotal>0);
                % Itotal = Itotal - min(Itotal(:));
                Itotal = Itotal ./ max(Itotal(:));
                Itotal = round(Itotal .* levels) / levels;

                [xc,yc,x2,y2]=MomentsWithThreshold(Itotal,threshold);
                if (drawing)
                    subplot(2,2,3);
                    imagesc(Itotal);
                    axis image;
                    colormap(gray);
                    hold on;
                    plot(xc,yc,'r*');
                end
                %                 dxval(MC) = dpix.*(xc+xoffsetsf-xc0);
                %                 dyval(MC) = dpix.*(yc-yc0);
                dxval(MC) = dpix.*(xc-xcIdeal);
                dyval(MC) = dpix.*(yc-ycIdeal);
            end;
```

```
            InoiseRMSVec(Nnoise,NpixSize,Noffset)=mean(InoiseRMS);
            IsignalRMSVec(NpixSize,Noffset)=RMS(Ipixelated);
            SNR(Nnoise) = RMS(Ipixelated)./mean(InoiseRMS);
            %             dx(NpixSize,Noffset,Nnoise) = mean(dxval);
            %             dy(NpixSize,Noffset,Nnoise) = mean(dyval);
            %             dxpix(NpixSize,Noffset,Nnoise) = mean(dxval)./dpix;
            %             dypix(NpixSize,Noffset,Nnoise) = mean(dyval)./dpix;
            dx(NpixSize,Noffset,Nnoise) = sqrt(mean(dxval.^2));
            dy(NpixSize,Noffset,Nnoise) = sqrt(mean(dyval.^2));
            dxpix(NpixSize,Noffset,Nnoise) = sqrt(mean(dxval.^2))./dpix;
            dypix(NpixSize,Noffset,Nnoise) = sqrt(mean(dyval.^2))./dpix;
            dxstd(NpixSize,Noffset,Nnoise) = std(dxval);
            dystd(NpixSize,Noffset,Nnoise) = std(dyval);
            dxstdpix(NpixSize,Noffset,Nnoise) = std(dxval)./dpix;
            dystdpix(NpixSize,Noffset,Nnoise) = std(dyval)./dpix;
        end;
        if (drawing)
            drawnow;
            pause(0.01); %(0.1);
        end;
    end;
end;
%profile off;
%profile viewer;

nf; errorbar(dx,dxstd)
xlabel('Pixel Size');
ylabel('Average Centroid Error');

% nf; errorbar(dx',dxstd')
% xlabel('SNR');
% ylabel('Average Centroid Error');

%% most processing
NpixelSizes = size(dx,1);
Noffsets = size(dx,2);
Nnoises = size(dx,3);

for ii=1:NpixelSizes;
    for jj=1:Nnoises;
        dx1(ii,jj) = mean(dx(ii,:,jj));
        dy1(ii,jj) = mean(dy(ii,:,jj));
        dxpix1(ii,jj) = mean(dxpix(ii,:,jj));
        dypix1(ii,jj) = mean(dypix(ii,:,jj));
        dxstd1(ii,jj) = std(dxstdpix(ii,:,jj));
        dystd1(ii,jj) = std(dystdpix(ii,:,jj));
    end;
end;

nf; plot(SNR,dxpix1);
nf; plot(dpixvec,dxpix1');

nf; imagesc(NnoiseVec,dpixvec,dx1);
colorbar;
```

```
%% different plotting
nf; imagesc(SNR,nxvec,dxpix1);
xlabel('SNR (RMS signal to RMS Noise)');
ylabel('Pixels Per Sub-Aperture (#)');
title('X-Axis (Scan direction) error');
colorbar; clabel('pixels');

% plot region <1/20 pixel
nf; imagesc(SNR,nxvec,dxpix1<0.05);
xlabel('SNR (RMS signal to RMS Noise)');
ylabel('Pixels Per Sub-Aperture (#)');
title('X-Axis (Scan direction) Region of 1/20th pixel Centroid Accuracy');
colorbar; clabel('pixels');

%plot absolute motion
nf; imagesc(SNR,nxvec,dx1);
xlabel('SNR (RMS signal to RMS Noise)');
ylabel('Pixels Per Sub-Aperture (#)');
title('Centroid Error in X-Axis (Scan direction) error');
colorbar; clabel('meters');

%different plotting y
nf; imagesc(SNR,nxvec,dy1);
xlabel('SNR (RMS signal to RMS Noise)');
ylabel('Pixels Per Sub-Aperture (#)');
title('Y-Axis (Non-Scan direction) error');
colorbar;

%different plotting slopes x2
nf; imagesc(SNR,nxvec,dx1./f);
xlabel('SNR (RMS signal to RMS Noise)');
ylabel('Pixels Per Sub-Aperture (#)');
title('Slope Error');
colorbar;

%% SNR plots
nf;
clear leg;
for ii=1:length(nxvec);
    plot(SNR,dxpix1(ii,:),getLineSpec(ii));
    hold on;
    leg{ii} = sprintf('%i pixels',nxvec(ii));
end;
ylabel('Average Centroid Error (pixels)');
xlabel('Signal to Noise Ratio');
legend(leg,'Location','EastOutside');

nf;
clear leg;
for ii=1:length(nxvec);
    semilogy(SNR,dxpix1(ii,:),getLineSpec(ii));
    hold on;
    leg{ii} = sprintf('%i pixels',nxvec(ii));
end;
ylabel('Average Centroid Error (pixels)');
```

```
xlabel('Signal to Noise Ratio');
legend(leg,'Location','EastOutside');

nf;
clear leg;
for ii=1:length(nxvec);

errorbar(SNR,10*log10(dxpix1(ii,:)),10*log10(dxstd1(ii,:)),getLineSpec(ii));
    hold on;
    leg{ii} = sprintf('%i pixels',nxvec(ii));
end;
ylabel('Average Centroid Error (pixels, 10*log10=dB)');
xlabel('Signal to Noise Ratio');
legend(leg,'Location','EastOutside');

%% pixel count plots
nf;
clear leg;
for ii=1:length(SNR);
    semilogy(nxvec,dxpix1(:,ii),getLineSpec(ii));
    hold on;
    leg{ii} = sprintf('SNR = %.1f',SNR(ii));
end;
ylabel('Average Centroid Error (pixels)');
xlabel('Pixels Across Sub-Aperture');
legend(leg,'Location','EastOutside');

% pixel count plots
nf;
clear leg;
for ii=1:length(SNR);
    errorbar(nxvec,dxpix1(:,ii),dxstd1(:,ii),getLineSpec(ii));
    hold on;
    leg{ii} = sprintf('SNR = %.1f',SNR(ii));
end;
ylabel('Average Centroid Error (pixels)');
xlabel('Pixels Across Sub-Aperture');
legend(leg,'Location','EastOutside');

%% relate the results to RMS WFE
%RMSWFE = 3 * dsub / f * dxRMS
Frecon=3;

%snr
nf;
clear leg;
for ii=1:length(nxvec);
    semilogy(SNR,dx1(ii,:)/f*D*Frecon,getLineSpec(ii));
    hold on;
    leg{ii} = sprintf('%i pixels',nxvec(ii));
end;
ylabel('Approximate RMS WFE (m)');
xlabel('Signal to Noise Ratio');
legend(leg,'Location','EastOutside');
```

```
% pixel count plots
nf;
clear leg;
for ii=1:length(SNR);
    semilogy(nxvec,dx1(:,ii)*Frecon*D/f,getLineSpec(ii));
    hold on;
    leg{ii} = sprintf('SNR = %.1f',SNR(ii));
end;
ylabel('Approximate RMS WFE (m)');
xlabel('Pixels Across Sub-Aperture');
legend(leg,'Location','EastOutside');

%% relate the results from RMS WFE to Strehl Ratio
%RMSWFE = 3 * dsub / f * dxRMS
% SR = exp(-phiRMS.^2)  in radians
Frecon=3;
kwave = 2.0*pi/wavelength;

%snr
nf;
clear leg;
for ii=1:length(nxvec);
    plot(SNR,exp(-1* (kwave*dx1(ii,:)/f*D*Frecon).^2),getLineSpec(ii));
    hold on;
    leg{ii} = sprintf('%i pixels',nxvec(ii));
end;
ylabel('Estimated Strehl Ratio');
xlabel('Signal to Noise Ratio');
legend(leg,'Location','EastOutside');

% pixel count plots
nf;
clear leg;
for ii=1:length(SNR);
    plot(nxvec,exp(-1*(kwave*dx1(:,ii)*Frecon*D/f).^2),getLineSpec(ii));
    hold on;
    leg{ii} = sprintf('SNR = %.1f',SNR(ii));
end;
ylabel('Estimated Strehl Ratio');
xlabel('Pixels Across Sub-Aperture');
legend(leg,'Location','EastOutside');

%% output the figures to ppt
if (ppt)
    SendAllFiguresToPowerPoint();
end;
```