LIGO SURF Progress Report 2: Particle Imaging and Analysis for Advanced LIGO

Sabrina Waller

Mentor: Calum Torrie

July 30, 2014

1. Introduction

I am continuing to develop a process where a Nikon D7100 with an AF Micro Nikkor 200mm 1:4 D lens can be remotely controlled to take images of the optics in Advanced LIGO. I am also writing code to process the images in MATLAB utilizing MATLAB's Image Processing Toolbox. The program counts the number of particles and measures their various sizes, providing accurate and immediate information about the contamination on the optics. An initial progress report on my work up to this point can be found in the following document: LIGO-T1400474-v1.

2. Progress

At the end of my first progress report, I was investigating purchasing a Nikon TC-200 or TC-201 Teleconverter to increase the resolution of the images and was beginning to experiment with MATLAB's image processing toolbox to discover which functions would be useful when writing a code to process the images and count the particles.

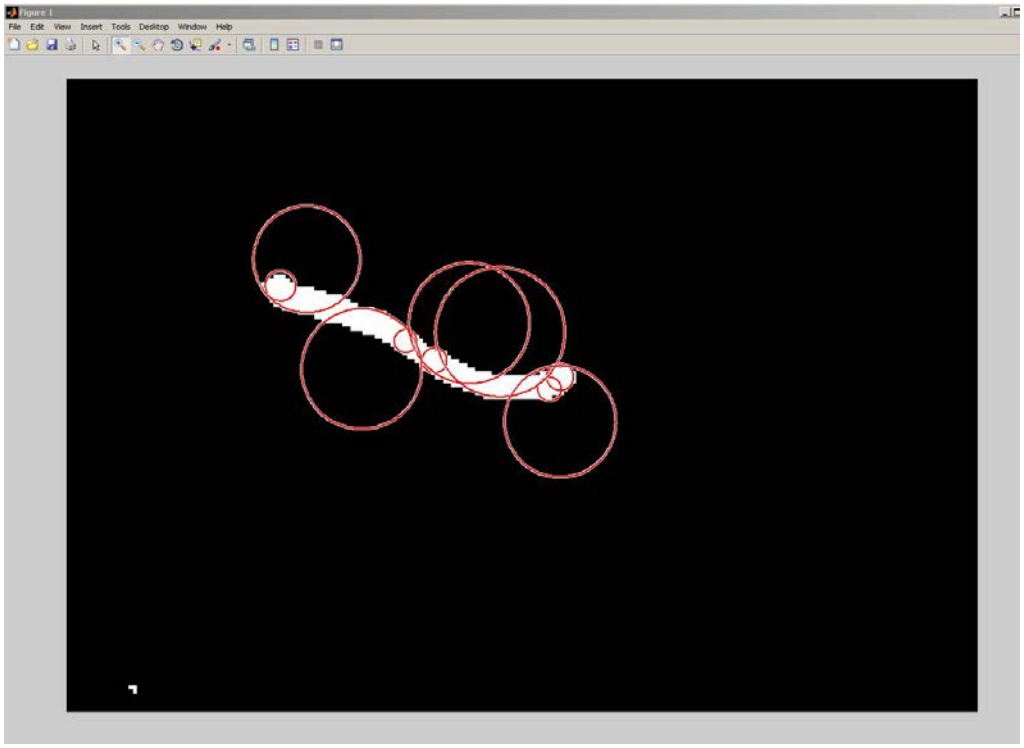
None of the camera shops in Pasadena carried a TC-200 or TC-201 Teleconverter, so we ended up purchasing a TC-201(2x) Teleconverter from Amazon. While waiting for the Teleconverter to arrive, I began experimenting in Matlab and soon realized that the functions necessary for detecting and measuring the particles only exist in MATLAB Image Processing Toolbox version 8.2 or greater. Once I downloaded version 9.0, the most recent version, I investigated the different functions in the toolbox and worked through the three following step by step MATLAB tutorials available on MathWorks.com: "Identifying Round Objects, "Detect and Measure Circular Objects on an Image," and "Detecting a Cell Using Image Segmentation." I spent several days familiarizing myself with MATLAB and its functions as I had never done any computer coding before and needed to understand how basic coding works in MATLAB before attempted to write my own code.

After I ran through these tutorials, I began experimenting with combining various functions in a code that could convert one of the images I took using the Nikon D7100 to binary, detect the particles in the image, circle the particles, and measure the area, perimeter, and diameter of the particles. The following is a code I wrote to try to detect and measure the particles using the imfindcircles function.

```
%Experimenting with matlab and lab picture 000161

%read image
rgb = imread('000161crop.jpg');
figure


% Convert image to grayscale
```

```
gray_image = rgb2gray(rgb);


%convert grayscale to binary
BW = im2bw(gray_image);


%tried smoothing the edges but approx half the particles disappeared
%seD = strel('diamond',1);
%BWfinal = imerode(BW,seD);
%BWfinal = imerode(BWfinal,seD);
%figure, imshow(BWfinal)


%detect circles
[center1, radius1] = imfindcircles(BW,[1 10],'ObjectPolarity', ...
    'bright','Sensitivity',0.95);
[center2, radius2] = imfindcircles(BW,[11 20],'ObjectPolarity', ...
    'bright','Sensitivity',0.95);

%outline 'Bright' particles with darker Circles of Different Color
imshow(BW);
hBright = viscircles(center1, radius1,'EdgeColor','r');
hBright = viscircles(center2, radius2,'EdgeColor','r');
```

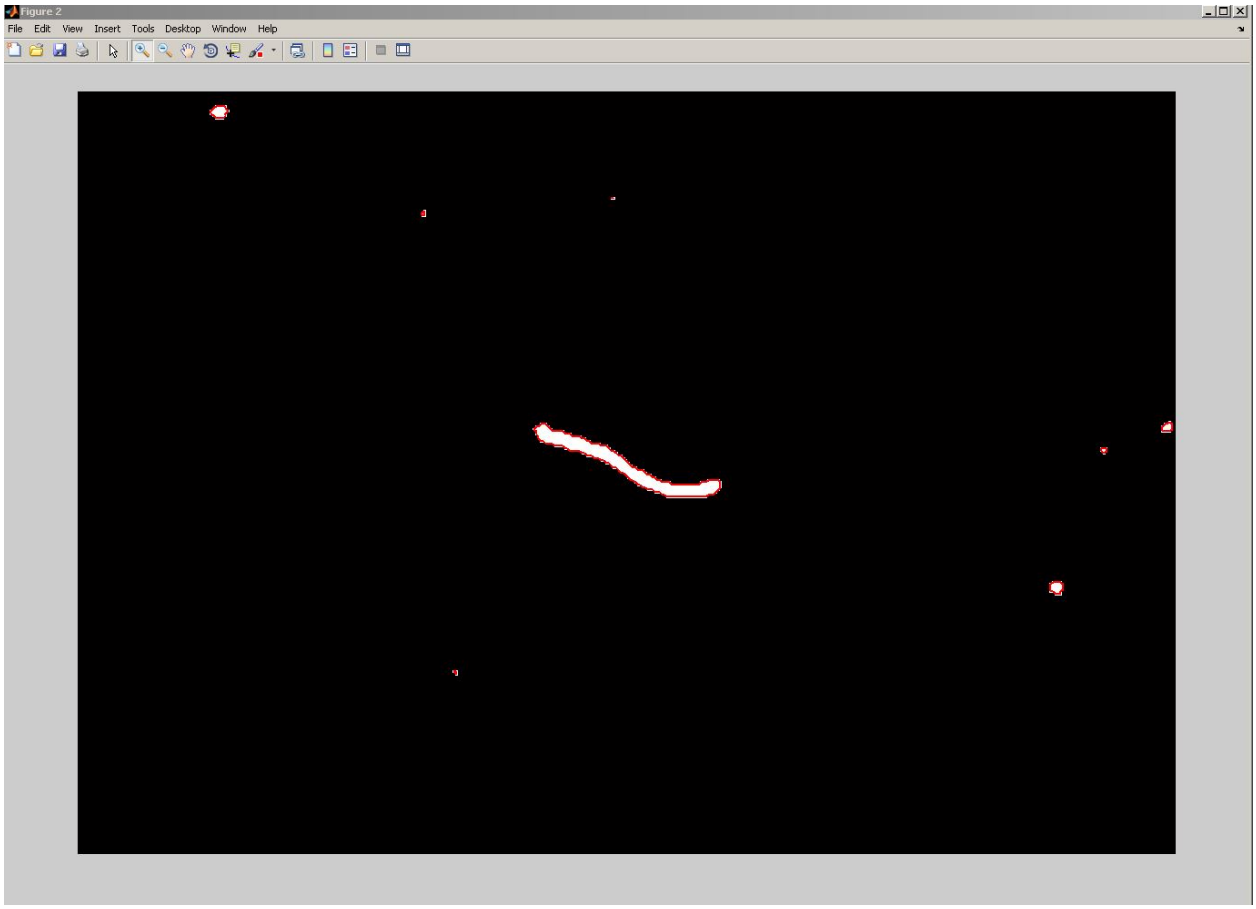
I quickly found that imfindcircles only detects perfect circles, and because the particles on the optic are not perfect circles, the function could not accurately detect them, as can be seen in the following close up of one of the particles on the processed image.



I tried increasing the sensitivity so that it would detect circular objects with higher eccentricities, but it continued to detect only perfect circles and even began circling particles that were not there. On the other hand, when I lowered the sensitivity, it was unable to detect a large portion of the particles. Therefore, I decided to see if I could use another function to outline the particles. I replaced the two sections of code that detected and outlined circles with the following code to outline the detected particles using the bwboundaries function.

```
%outline
[B,L] = bwboundaries(bw, 'noholes');
% figure, imshow(bw);
for k = 1:length(B)
    boundary = B{k};
    plot(boundary(:,2), boundary(:,1), 'r', 'LineWidth', 2)
end
```

MATLAB was able to accurately detect and outline the particles on the image. This success is evident in the following image, which is of the same particle that imfindcircles was unable to detect.

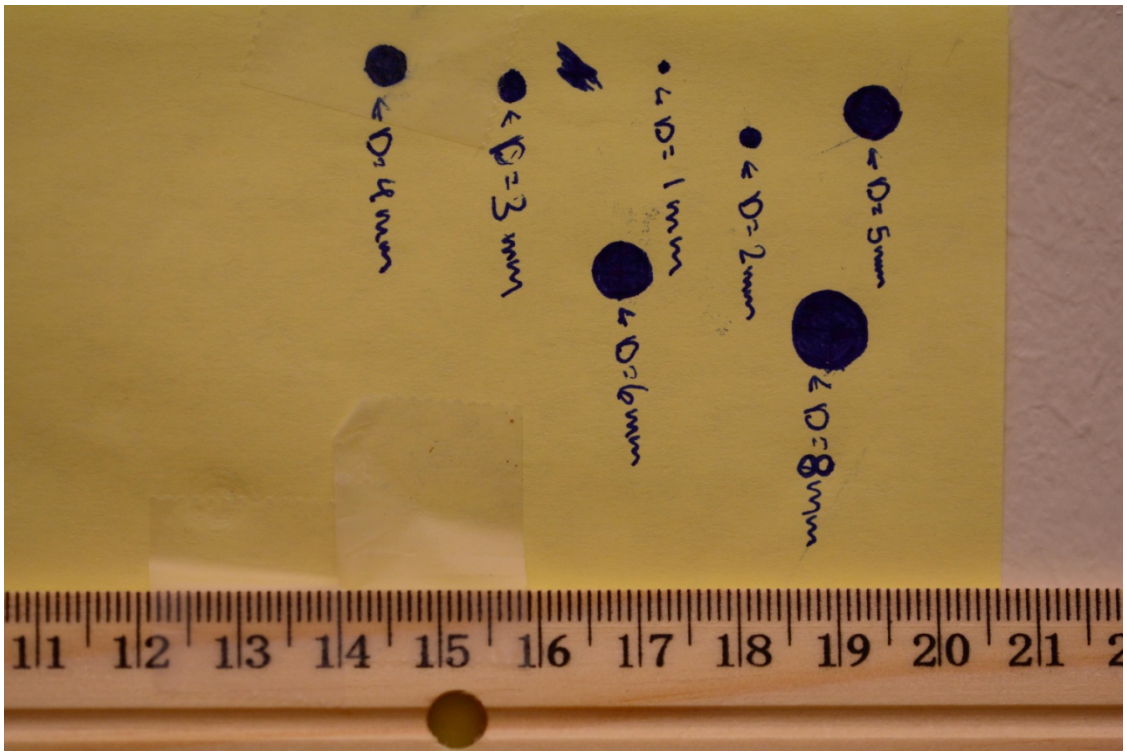


With outlines around the detected particles, I can see which particles MATLAB detects, and therefore, catch any errors in detection that may occur, such as if it misses particles, detects ones that do not exist, or under or over estimates their sizes.

I knew which particles MATLAB detected and how accurately it detected their edges, but I still needed a way to find the measurements of the detected particles. I discovered the function regionprops, which measures both the area and perimeter of connected components in the image. MATLAB displays the answers in terms of pixels, so I wrote a conversion to meters into the code, utilizing the pixel/meter ratio I calculated earlier when using Toupview. I explain the process of how I found this ratio in my first progress report. To check that the pixel/meter ratio was the same in MATLAB as Toupview, I loaded the same image of the ruler I used to calculate the ratio in Toupview and used the imdistline function to create a distance line. I used the distance line to measure 40mm and discovered that this distance was equivalent to 2138.08 pixels. From this information, I calculated that the ratio was 53452 Pixels/Meter, which is close to the 53544 Pixels/Meter ratio I calculated in Toupview, the slight difference due to human inability to draw a precisely 40mm line in pixels. I used this new ratio to convert the area and perimeter measurements from pixels to microns and millimeters. Once I had these measurements, I calculated what the particles' diameters would be if the particles were perfect circles. The following section of code measures the particles and performs these conversions on them.

```
%find the areas and perimeters of the objects detected in pixels
D1 = regionprops(bw, 'area');
D2 = regionprops (bw, 'perimeter');
%answers in structure array form
%will sometimes say perimeter=0 because can't measure when particle is only one pixel

%converting from structure array to matrix form
areatemp = cell2mat(struct2cell(D1)) ;
perimtemp = cell2mat(struct2cell(D2)) ;

%finding areas of particles in mm^2 and microns^2
area_mm = (areatemp/2857116304)*1e6; %pixels to mm^2
area_microns = (areatemp/2857116304)*1e12; % pixels to microns^2

%finding perimeter of particles in mm and microns
perim_mm = (perimtemp/53452)*1e3; %pixels to mm
perim_microns = (perimtemp/53452)*1e6; %pixels to microns

%finding radii assuming particles are perfect circles
radius_mm = ((area_mm)/pi).^(1/2);
radius_microns = ((area_microns)/pi).^(1/2);

%finding diameter assuming particles are perfect circles
diameter_mm = radius_mm.*2;
diameter_microns = radius_microns.*2;
```

I double checked the ratio by running the same test image with particles of known size through MATLAB that I ran through Toupview.



Test Image of Particles of Known Size

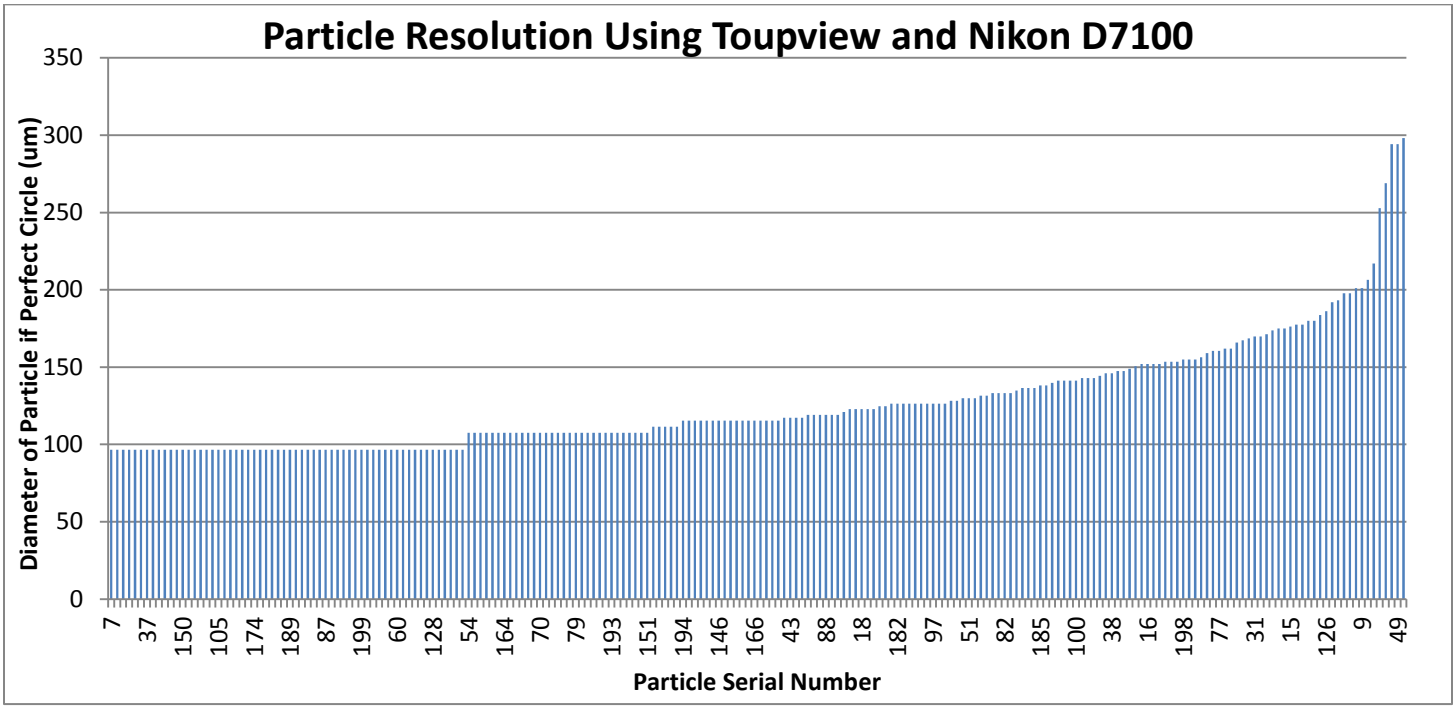
I used the imdistline function to measure the diameters of the circles in pixels and used the pixel/meter ratio I calculated to determine their diameters and areas in millimeters. The following chart shows the known dimensions as well as the dimensions according to Toupview and MATLAB.

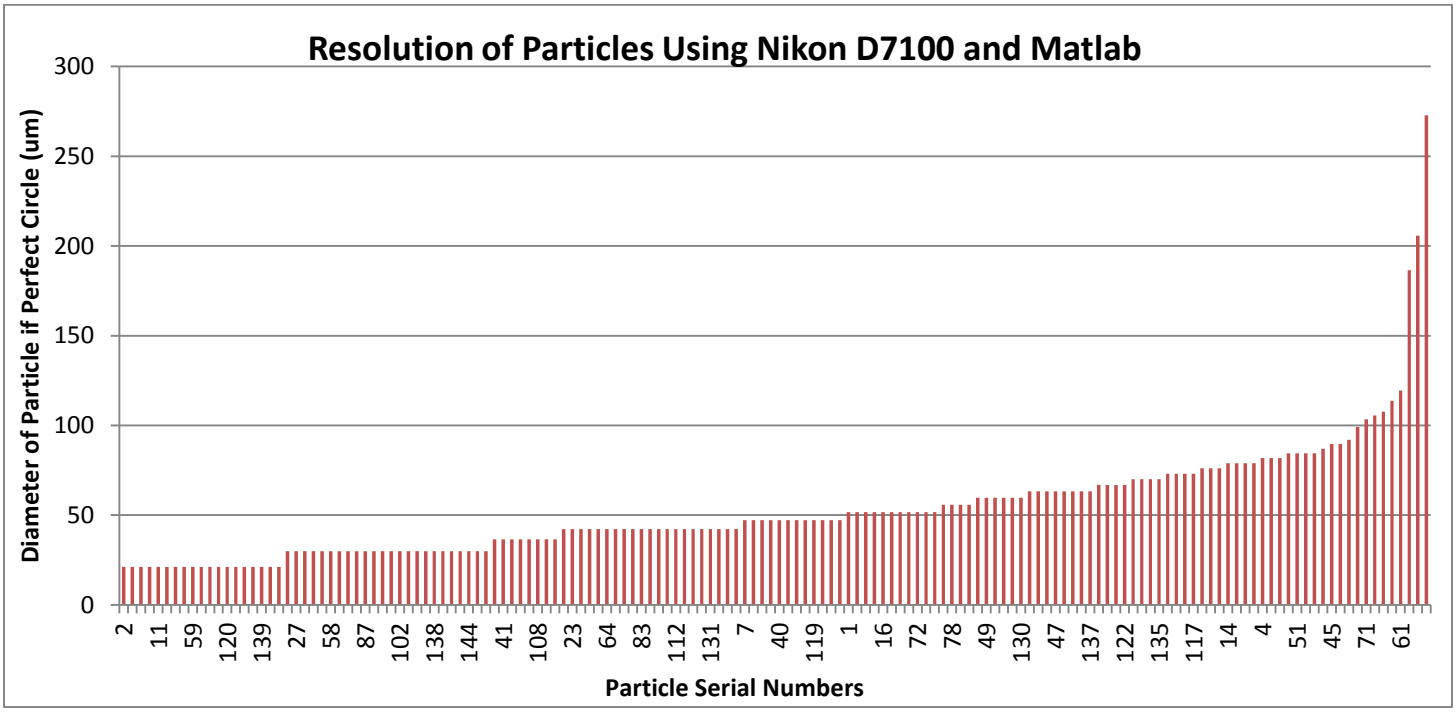
Hand Calculated Diameter (mm)	Matlab Diameter (pixels)	Matlab Diameter (mm)	ToupView Diameter (mm)	Hand Calculated Area (mm^2)	Matlab Area (mm^2)	ToupView Area (mm^2)
1	64.5	1.206690114	1.277009192	0.785398	1.143619077	1.28079
2	117	2.188879743	2.225262706	3.14159	3.762995382	3.88913
3	156.75	2.932537604	3.089575516	7.06858	6.754248903	7.497
4	217.88	4.076180498	4.259818917	12.56637	13.04958484	14.25188
5	294.75	5.514293198	5.595062155	19.63495	23.88193926	24.58667
6	325.88	6.096684876	5.999999588	27.9955	29.19290925	28.27433
8	420.75	7.871548305	7.999999804	49.782	48.66426979	50.26548

Just like Toupview, while the smaller measurements seem highly inaccurate, the circles were hand drawn and on such a small scale that there is a large margin for human error. Additionally, when I measured the 5 mm circle and the 4 mm circle again, the calculated diameters were more accurate than my initial measurements because the paper soaked up and spread the ink from the pen. In this case, the 6 mm and 8 mm circle data yield the most accurate results due to the decreased human error and higher precision from their increased size. Considering the human error involved, the

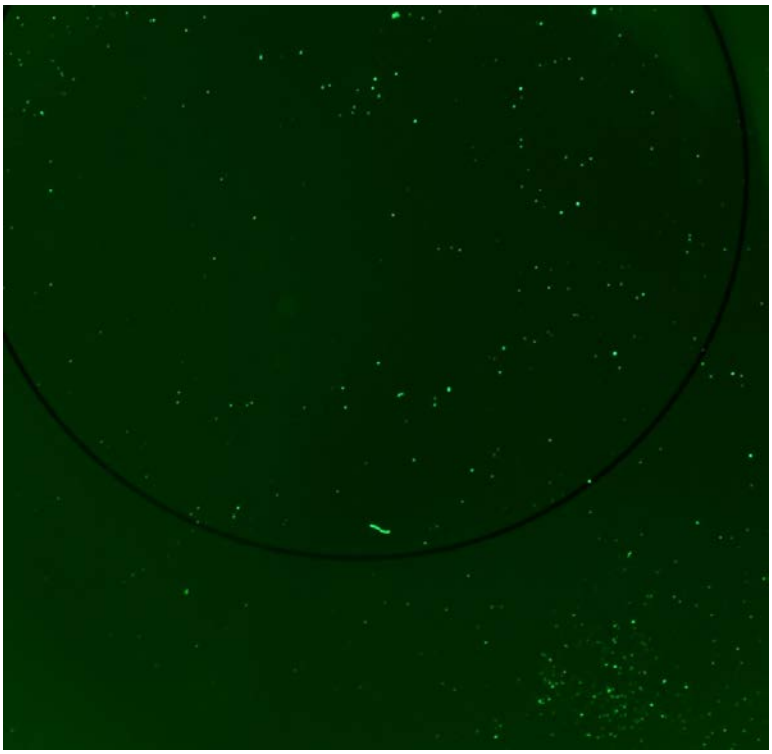
differences between the diameter and area calculated using MATLAB , the ones calculated in Toupview, and the known ones based on hand measurements are so slight as to be negligible. Therefore, the results confirm the accuracy of the 53452 pixel/meter ratio in MATLAB for the Nikon D7100 with an AF Micro Nikkor 200mm 1:4 D lens 40 inches away from the optic.

Next, I ran an actual image of the optic through both Toupview and MATLAB to see if their results were comparable or if one program outperformed the other. The full Data Sheet of the results can be found in the following document: LIGO-T1400494-v1. As can be seen in the graphs below, the smallest particles Toupview resolved had a diameter of 96.57259349 microns. Approximately one third of the particles are 96.57259349 um, which reveals that Toupview rounds any particles below or around this size to this lower limit, which makes the size measurements for the smaller particles highly inaccurate. On the other hand, MATLAB resolved particles that were far smaller, down to a diameter of 21.11013932 microns, which demonstrates that MATLAB can resolve particles that are over 70 microns smaller than Toupview. However, while MATLAB only detected 152 particles with the largest particle reaching a diameter of 272.8031214 microns, Toupview detected 218 with the largest particle reaching a diameter of 298.0294938 microns; MATLAB erased and shrunk particles during processing.

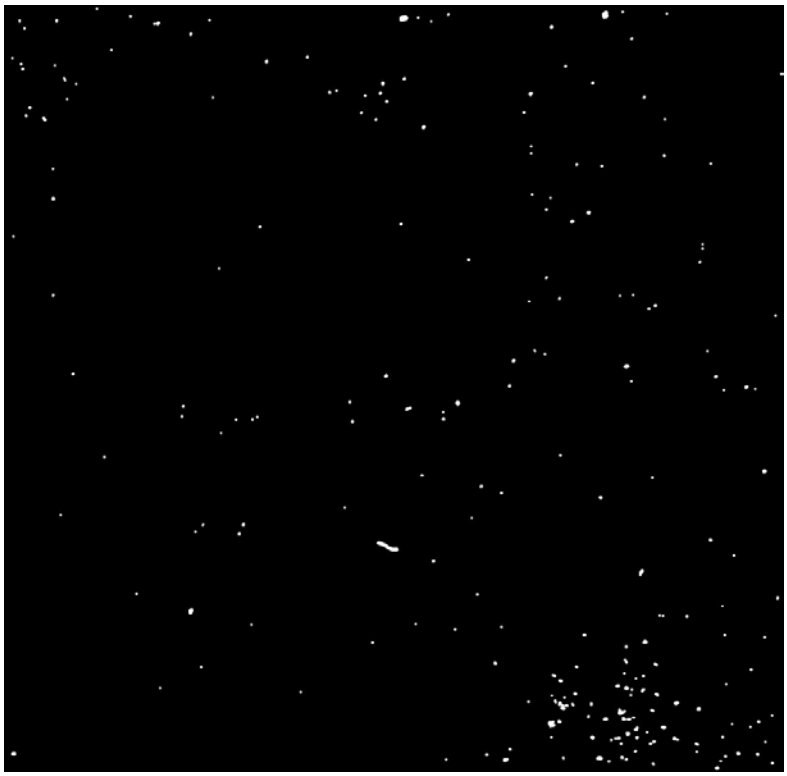




By examining the processed image, I discovered that this error is due to MATLAB's use of a default setting when it converts images to binary. The automatic settings create inaccurate results because each image requires unique settings. Detecting dust particles requires a high level of accuracy, so the settings have to be precise. If the threshold is too high, MATLAB will read certain particles or sections of the particles as part of the background and color them black, effectively shrinking them or erasing them from the image. However, if the threshold is too low, MATLAB will read brighter parts of the background as particles and detect ghost particles that do not actually exist. As can be seen in the following images, the hand thresholded binary image from Toupview is far more accurate in terms of the size and number of particles than the automatically thresholded image from MATLAB. The MATLAB binary image is missing a large number of particles and the particles that remain are far smaller than they were in the original image.



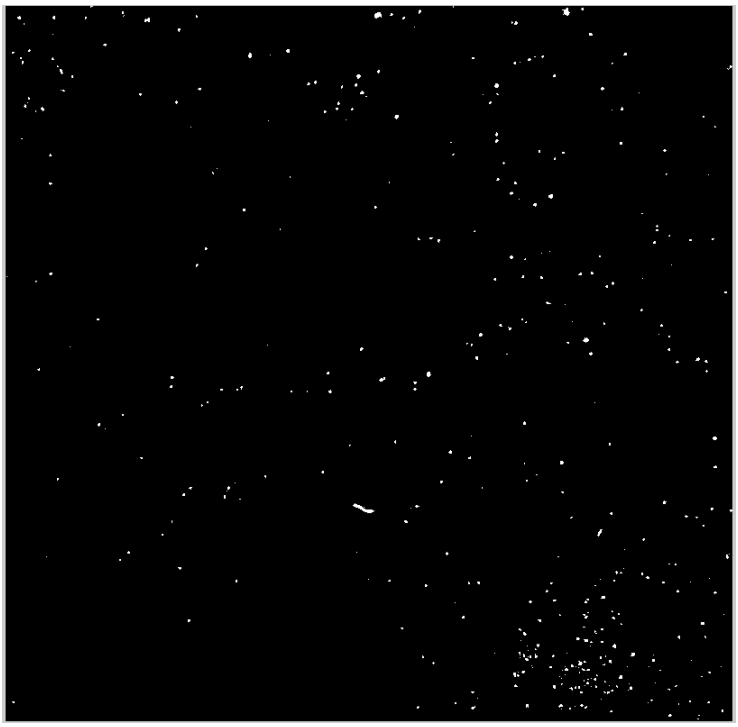
Original Image



Toupview Hand Thresholded Binary Image

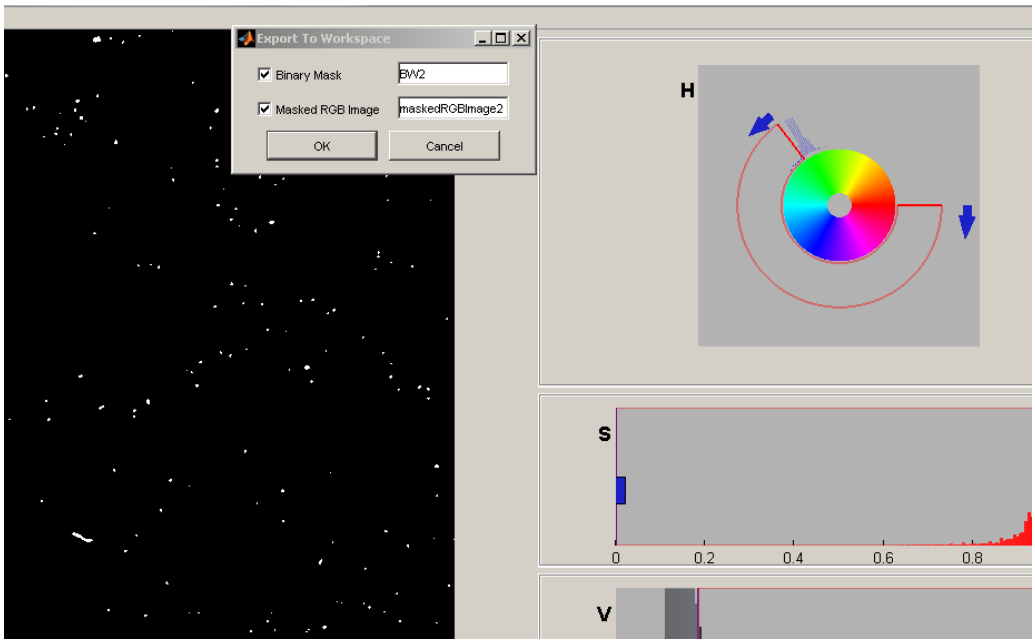
MATLAB Automatically Thresholded Binary Image

Having isolated the source of the problem as the automatic conversion to binary, I then had to find a way to write a control into the code that would allow the operator to adjust the threshold by hand. After searching for several hours, I discovered that Version 9 of MATLAB image processing toolbox, the most recent version, has a preexisting color thresholder app. I ran the code once to generate a binary image, clicked on the app, and imported the image by clicking on the green arrow that says “import” and then clicking on “import from workspace.” I was then able adjust the HSV color space by hand to threshold the image. To properly adjust the image, I followed a Mathworks tutorial titled “Image Segmentation Using the Color Thresholder App.” The following is the resulting binary image:



Matlab Hand Thresholded Binary Image

I then exported the image, inserted the name of the new image into the line of code following the conversion to binary, the line that fills in holes in the particles, and ran the program a second time to measure the particles on the new binary image.

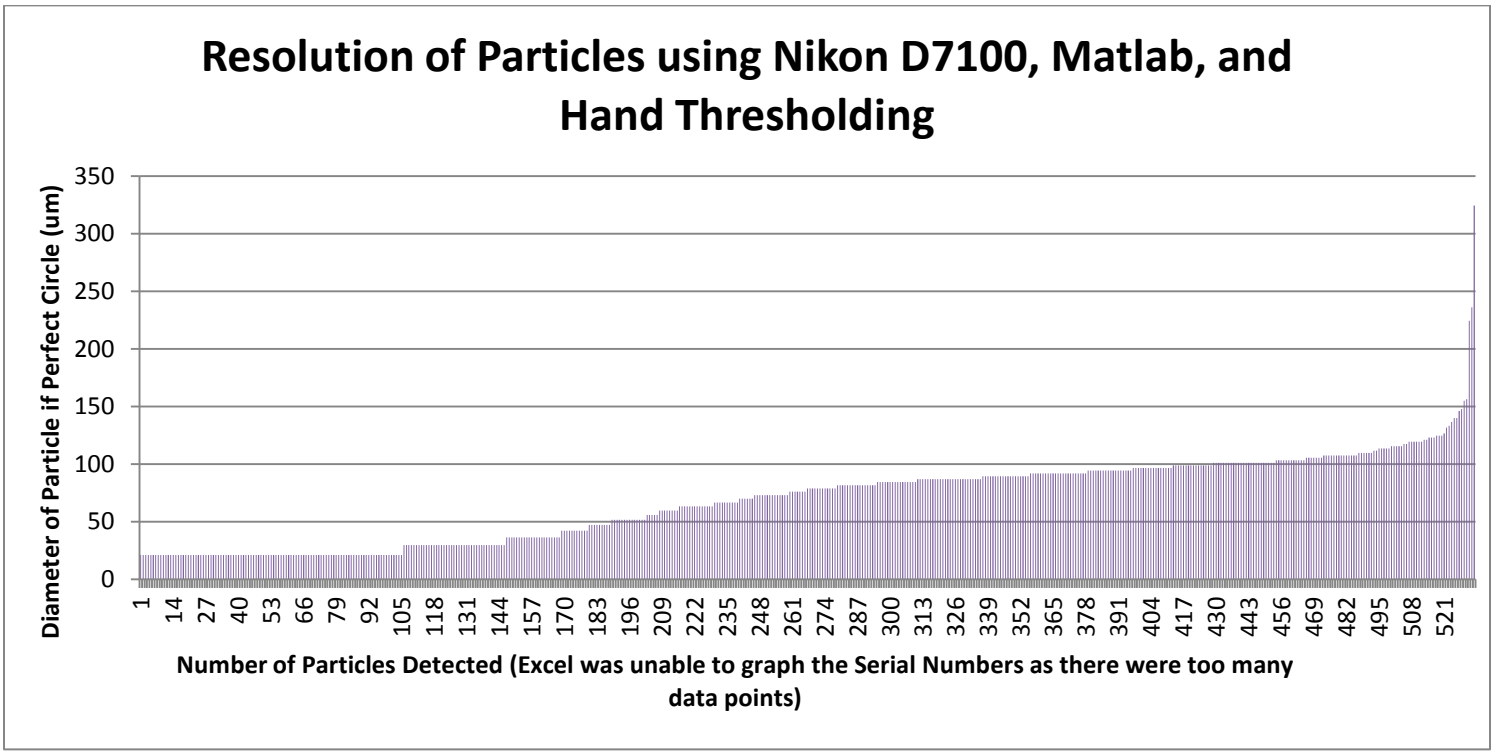


Exporting the new image from the Color Thresholder App. "BW2" is the name of the new image.

```
%fill in holes
bw = imfill(BW2,'holes');
```

Inserting the name of the image into the next line of code.

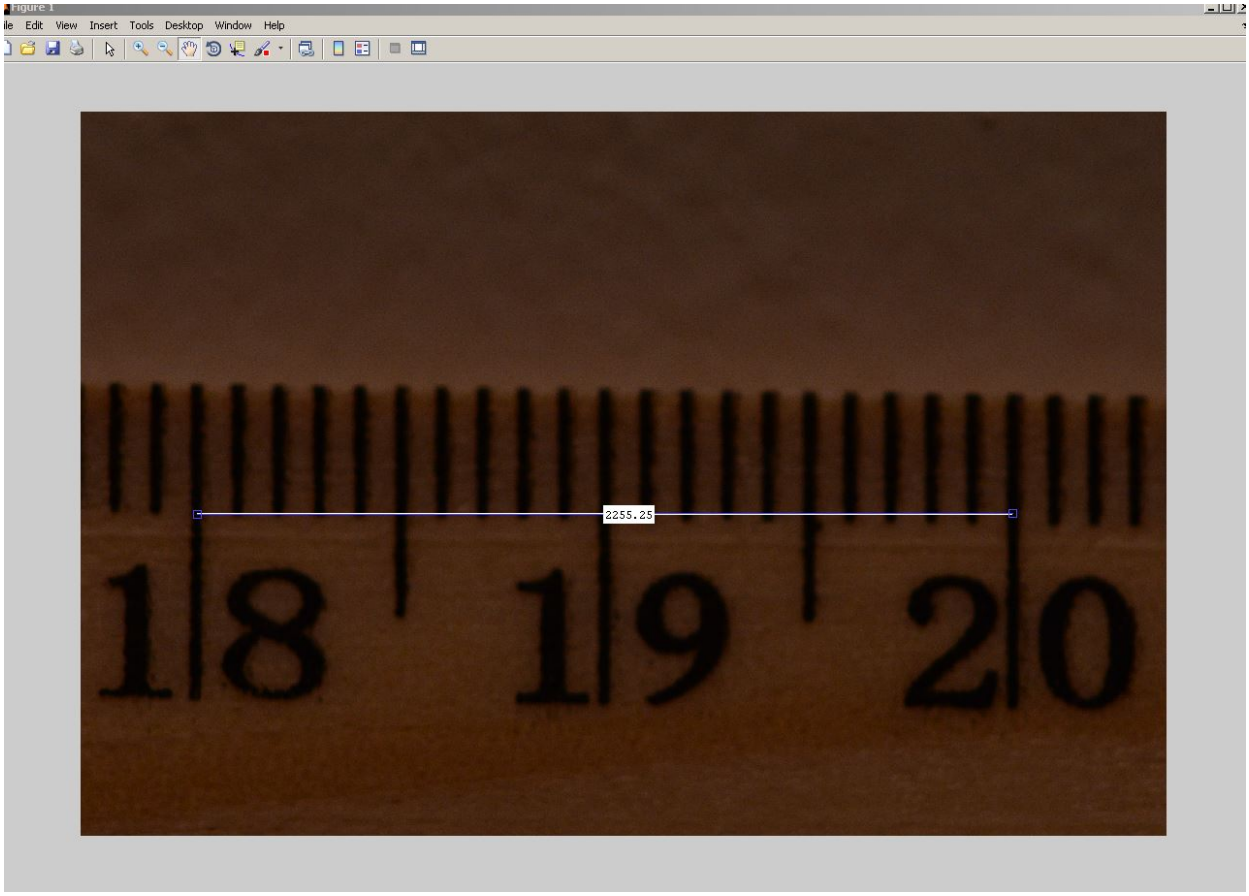
With the new hand thresholded image, MATLAB detected 533 particles, ranging in size from a diameter of 21.11013932 microns to 324.3001138 microns. This data fits with the data collected from Toupview because MATLAB can detect particles far smaller than Toupview, and thus, detects more particles than Toupview. Furthermore, when I compared the Toupview image, the hand thresholded MATLAB image, and the original image, I found that the MATLAB image is more accurately thresholded than the Toupview one. The slight over thresholding of the Toupview image accounts for the increase in particles and particle size, a growth best seen by the 26 micron increase in the diameter of the largest particle from Toupview to MATLAB. The Bar graph showing the resolution of the particles can be found below and in document LIGO-T1400494-v1 along with the full Data Sheet of the results.



It is evident from the graph of the results that the smallest size MATLAB can detect is a particle with a diameter of 21.11013932 microns. Additionally, because the pixel size is 1/53452 of a meter, the area of each pixel is 350.0032528 um^2 and thus, the area measurements increase in jumps up of 350.0032528 um^2 , which lowers the accuracy of the results. Therefore, while the resolution of particles using the Nikon D7100 in MATLAB is superior to the resolution in Toupview, the resolution is still not optimum. To increase the resolution, I turned to the recently arrived Nikon TC-201 teleconverter. Because the teleconverter is designed to double the magnification of the image, it would, in theory, double the resolution of the image.

I repeated the steps I went through without the teleconverter to determine the new image size and calculate the new resolution. I taped a ruler to a white wall and took pictures of it from a 40 inch distance to accurately recreate the distance the camera will be from the optic when on site. I checked the images and found that a single image covers an area of 53 mm x 35 mm.

I then calculated the image resolution using the same process I described earlier and found that 20 mm is equivalent to 2255.25 pixels, which makes the pixel to meter ratio for the Nikon D7100 with the 2x Teleconverter 1/112762.5 Pixels/Meter. This ratio is 2.1096 times greater than the pixel/meter ratio for the camera without the teleconverter, which confirms that the teleconverter doubles the resolution.



Calculating Pixel to Meter Ratio for Nikon D7100 and 2x Teleconverter

When I went to take a picture of the optic, I discovered that the addition of the teleconverter darkens the image considerably. I retested the settings and after comparing all my test images, I discovered that the best, most consistent setting is as follows:

Aperture: f-stop 5.6 (With the Teleconverter, the aperture reads as f0 in ControlMyNikon and the program no longer allows the operator to change the aperture remotely. Therefore, the operator has to change it on the camera itself.)

Shutter Speed: 1/40

ISO: 200

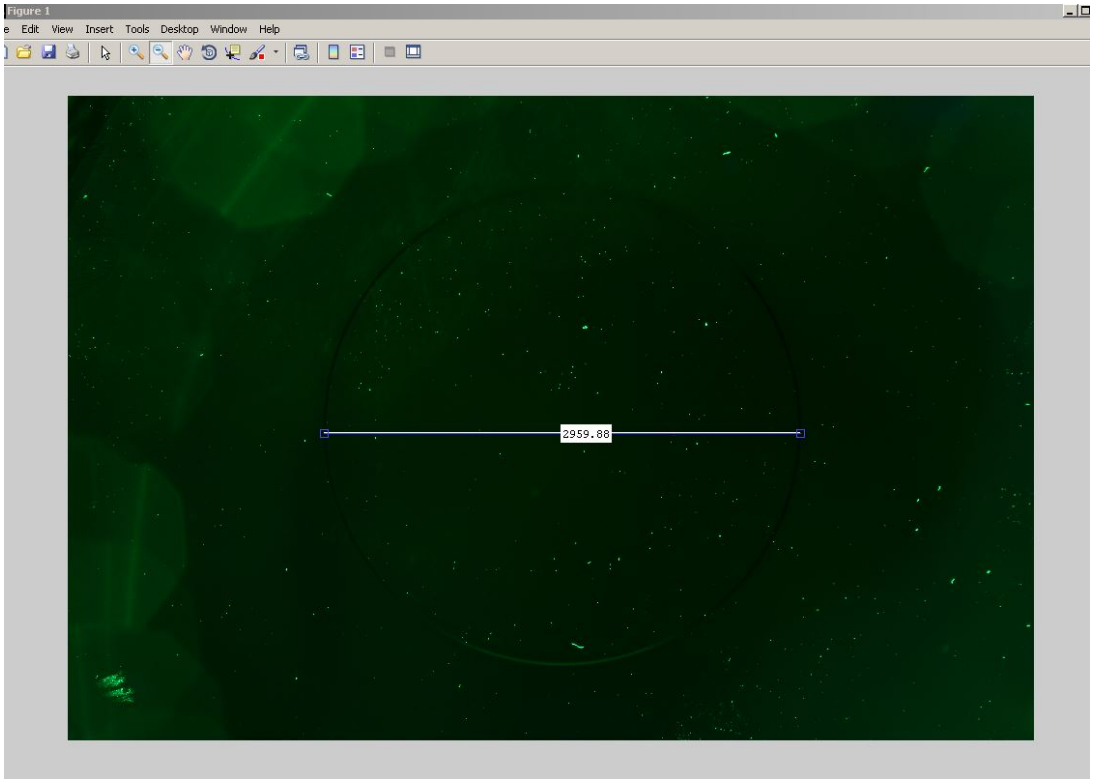
White Balance: Fluorescent

Image Quality: Fine

Picture Control: Vivid

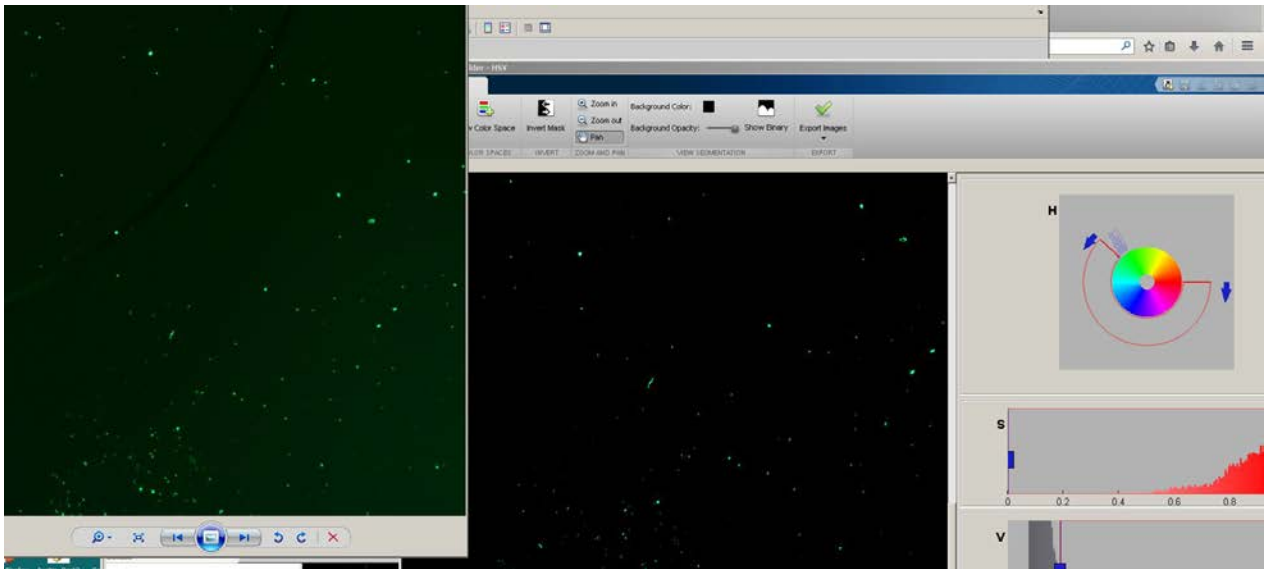
Green LEDs: Anywhere within 28 to 31 Volt bracket

For further confirmation of the new ratio, I measured the diameter of the inner circle on the optic in my new image, which I know to be around 27 millimeters. I used the 1/112762.5 ratio to calculate the distance as 26.2 mm across, which confirms that the pixel to meter ratio is accurate. The slight variation is due to human inability to draw a distance line in the exact correct number of pixels, not knowing exactly where the diameter of the circle was on the image, and knowing only the approximate size of the ring.



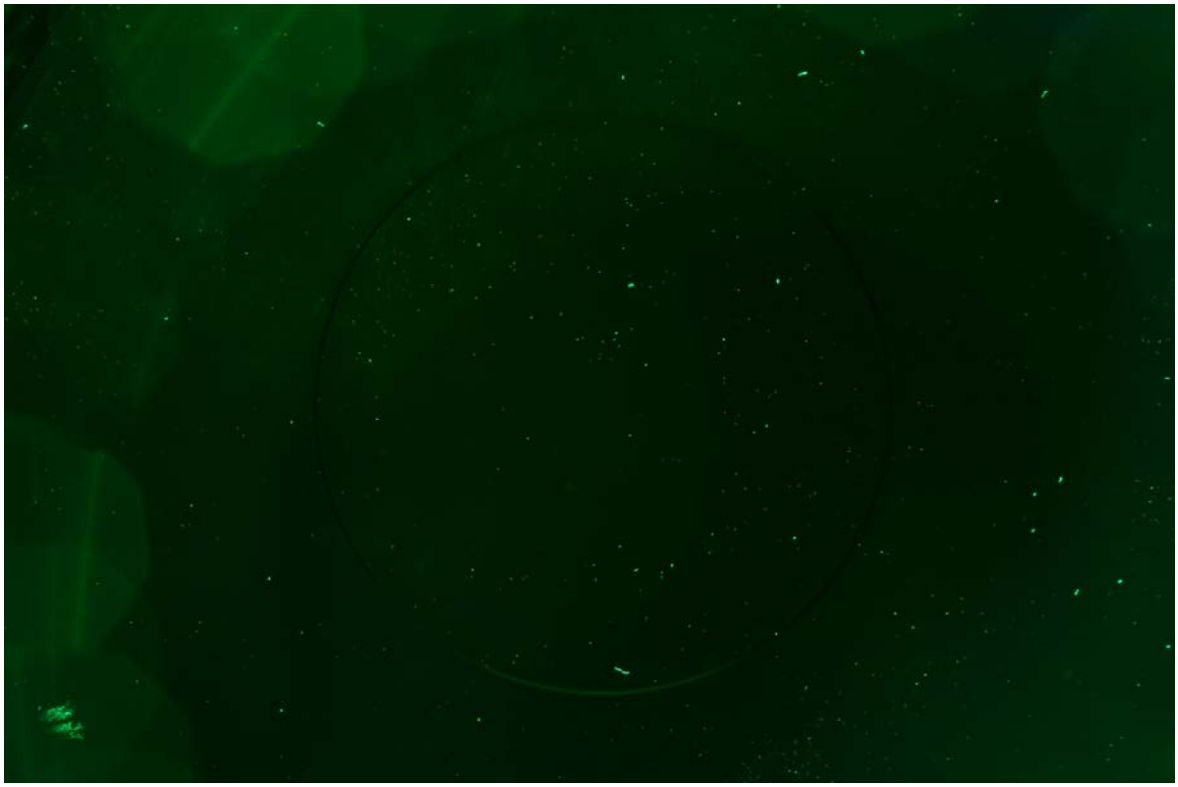
Measuring the Circle on the Optic to Confirm the Resolution Ratio

I then ran the image through the code and hand thresholded it. As I worked with the thresholder app, I discovered that the best way to accurately threshold the images is to open the original image and put it next to the one being thresholded so that it is easy to see when particles disappear or ghost particles appear.

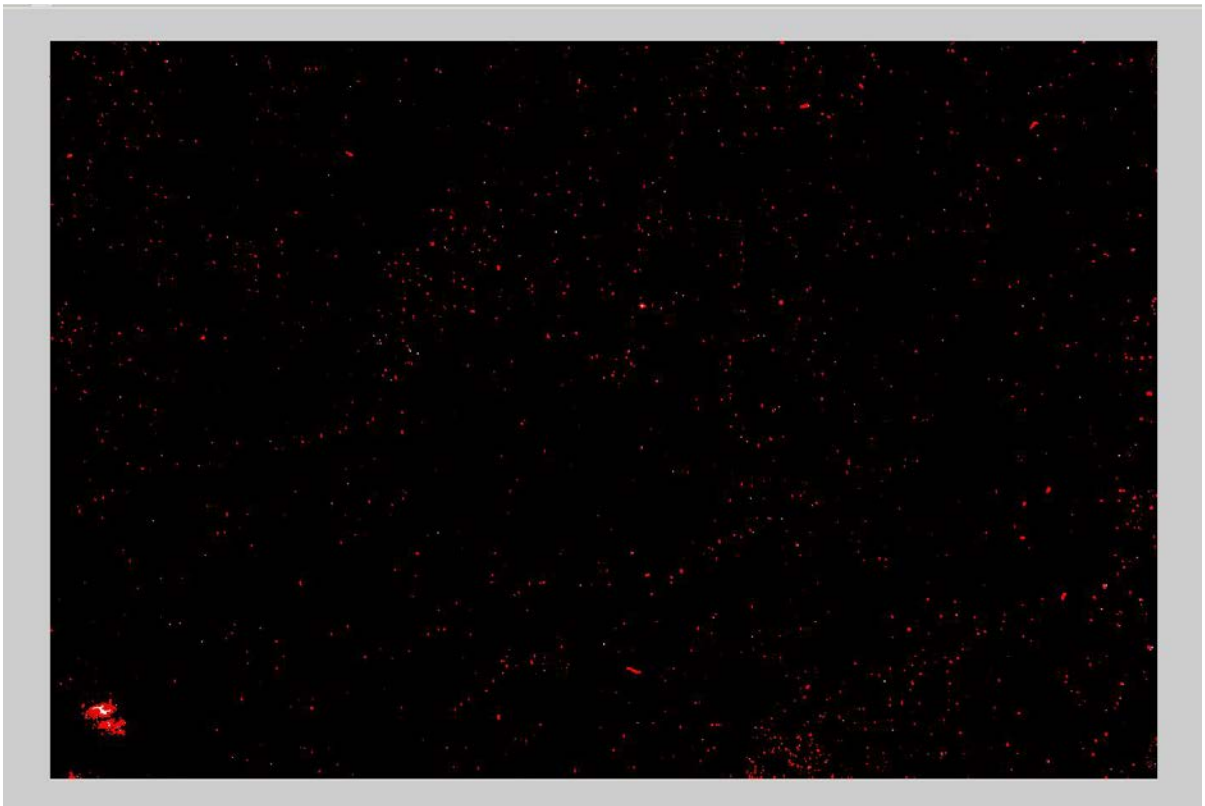


Thresholding the Image While Comparing it to the Original

I inserted the name of the new image into the code and ran it a second time. Before examining the measurements, I compared the original image and the processed one to check that the particles on both images matched and found that they did.

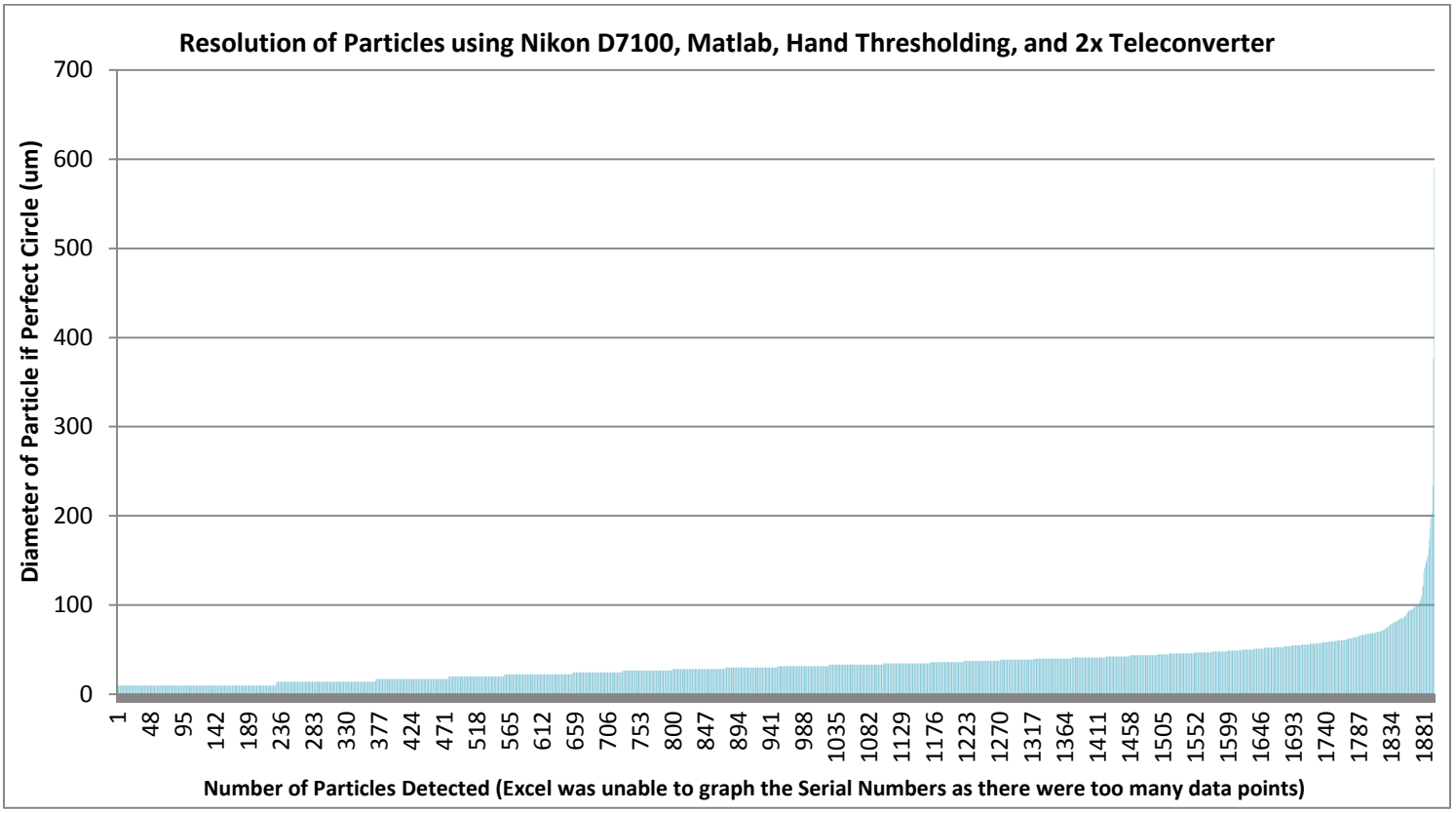


Original Image Taken With Nikon D7100 and 2x Teleconverter



Hand Thresholded Image with Outlined Particles Taken With Nikon D7100 and 2x Teleconverter

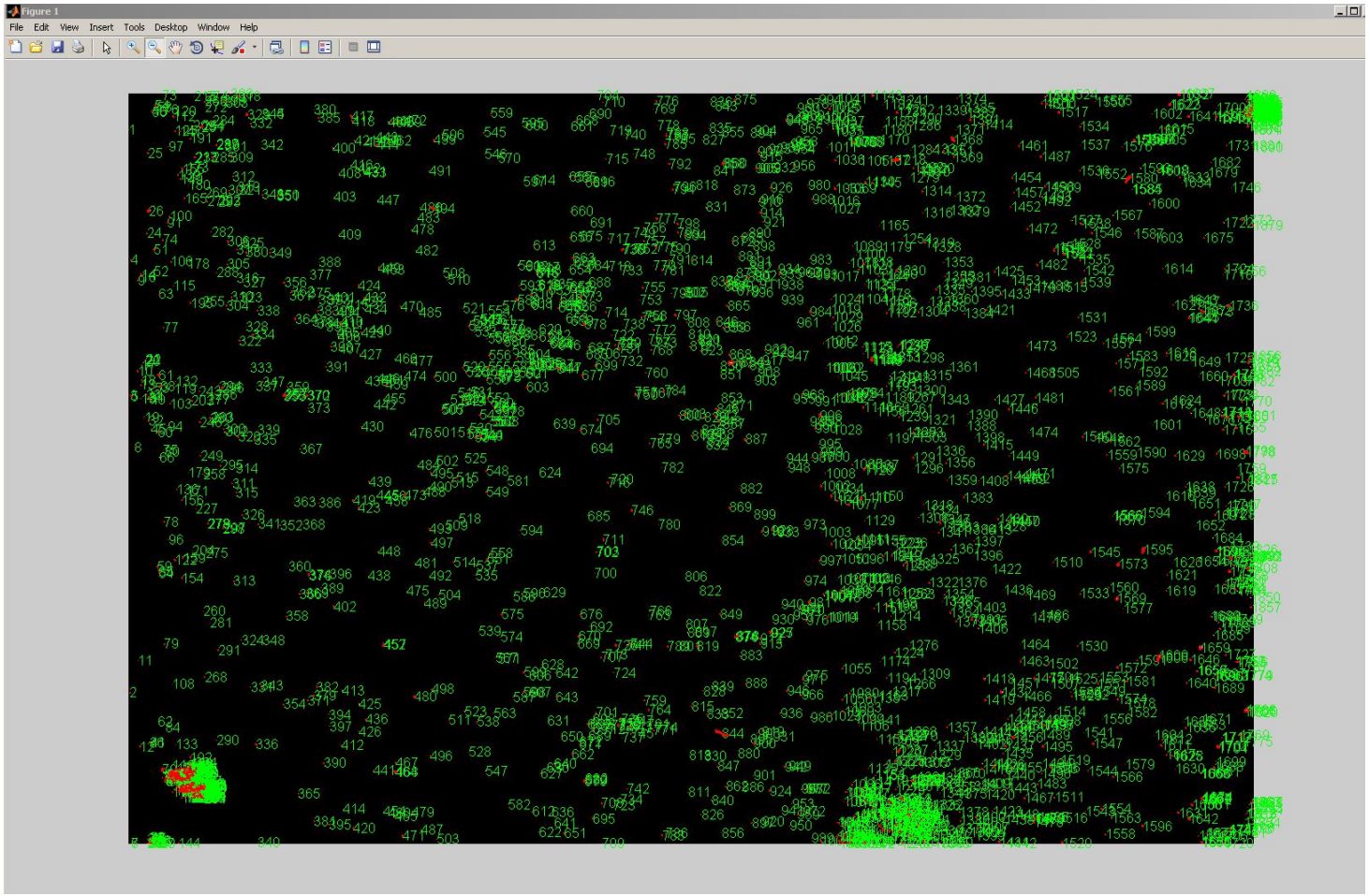
MATLAB detected 1896 particles on the image. The smallest particle had a diameter of 10.0066881 microns and the particles increased in area in steps of 78.64490791 um^2 up to the largest particle, which had a diameter of 591.326689 microns. The addition of the teleconverter made it possible for MATLAB to resolve and detect particles just over 10 microns in diameter smaller than it had been able to with just the Camera and Nikkor Lens. The teleconverter also decreased the pixel size from 1/53452 of a meter to 1/112762.5 of a meter, which in turn, significantly decreased the jumps between the particle sizes and increased the accuracy of the measurements. The Bar Graph of the resolution of the particles can be found below and in document LIGO-T1400491-v1 along with the full Data Sheet of the results.



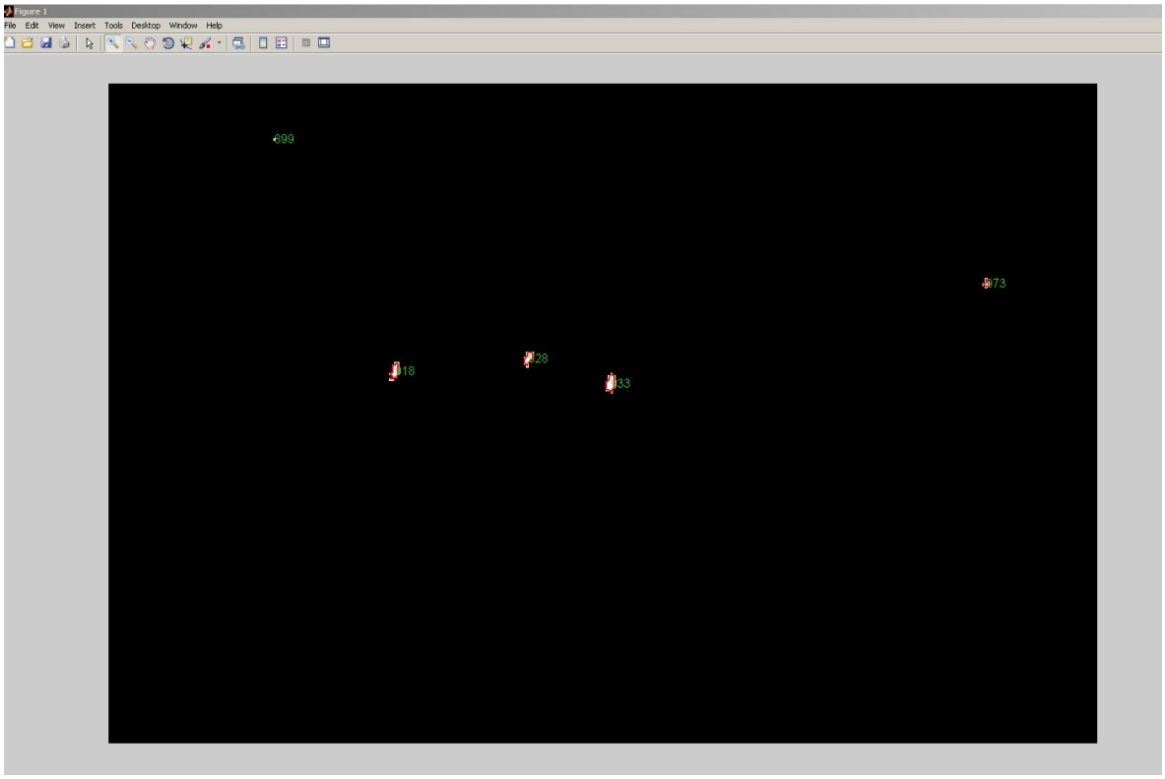
While the data indicates that MATLAB is able to resolve and detect particles over a large range, I did not know for certain how well the system resolves particles at the various sizes. In order to check that the system does not have any problems resolving particles of specific sizes and see what the different size particles look like, I had to examine particles of these different sizes on the image. However, while MATLAB does detect and measure the particles in a particular order, thus giving each particle an identifying number, it does not formally assign or display serial numbers for the particles. Therefore, I could not find a way to display the serial numbers on the image and did not know which particles matched which measurements. After trying for several days to write a piece of code that would assign and display serial numbers for the various particles, I finally succeeded by measuring the x,y, coordinates of the various particles using the centroid function and writing code that goes through and assigns each of the particles a number. The code then displays this number on the image and changes the text color from the default color of black to green. The serial numbers match the order of the particle measurements because centroid is a regionprops function just like the functions that measure the particles, and all of the regionprops functions measure the particles by following the same pattern based on the x,y coordinates of the particles. The following is the section of code that numbers the particles and displays the numbers on the image.

```
%number the particles
s = regionprops(L, 'centroid');
centroids = cat(1, s.Centroid)

for K=1: length(centroids)
```

```
    x=centroids(K,1);
    y=centroids(K,2);
    a=num2str(K);
    text(x,y,sprintf('%d',K),'color','g','Fontsize',12)
%     text(x,y,[sprintf('%d',i),'color',x(k),'Fontsize',8

end
```



Processed Image with Numbering



Up close Image of Numbered Particles

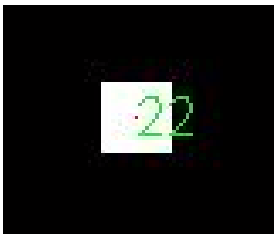
With the inclusion of this section, the final code is as follows:

```
%read image
rgb = imread('000200 (teleconverter).jpg');


% open color thesholder app
colorThresholder(rgb)


%fill in holes
bw = imfill(BW9,'holes');
imshow(bw)


hold on

%outline
[B,L] = bwboundaries(bw, 'noholes');
% figure, imshow(bw);
for k = 1:length(B)
    boundary = B{k};
    plot(boundary(:,2), boundary(:,1), 'r', 'LineWidth', 2)
end
```

```
%number the particles
s = regionprops(L, 'centroid');
centroids = cat(1, s.Centroid)

for K=1: length(centroids)
    x=centroids(K,1);
    y=centroids(K,2);
    a=num2str(K);
    text(x,y,sprintf('%d',K),'color','g','Fontsize',12)
%     text(x,y,[sprintf('%d',i),'color',x(k),'Fontsize',8

end

hold off


%find the areas and perimeters of the objects detected in pixels
D1 = regionprops(bw, 'area');
D2 = regionprops (bw, 'perimeter');
%answers in structure array form
%will sometimes say perimeter=0 because can't measure when outer pixels don't
%directly connect on image

%converting from structure array to matrix form
areatemp = cell2mat(struct2cell(D1)) ;
perimtemp = cell2mat(struct2cell(D2)) ;

%finding areas of particles in mm^2 and microns^2
area_mm = (areatemp/12715381410)*1e6; %pixels to mm^2
area_microns = (areatemp/12715381410)*1e12; % pixels to microns^2

%finding perimeter of particles in mm and microns
perim_mm = (perimtemp/112762.5)*1e3; %pixels to mm
perim_microns = (perimtemp/112762.5)*1e6; %pixels to microns

%finding radii assuming particles are perfect circles
radius_mm = ((area_mm)/pi).^(1/2);
radius_microns = ((area_microns)/pi).^(1/2);

%finding diameter assuming particles are perfect circles
diameter_mm = radius_mm.*2;
diameter_microns = radius_microns.*2;
```

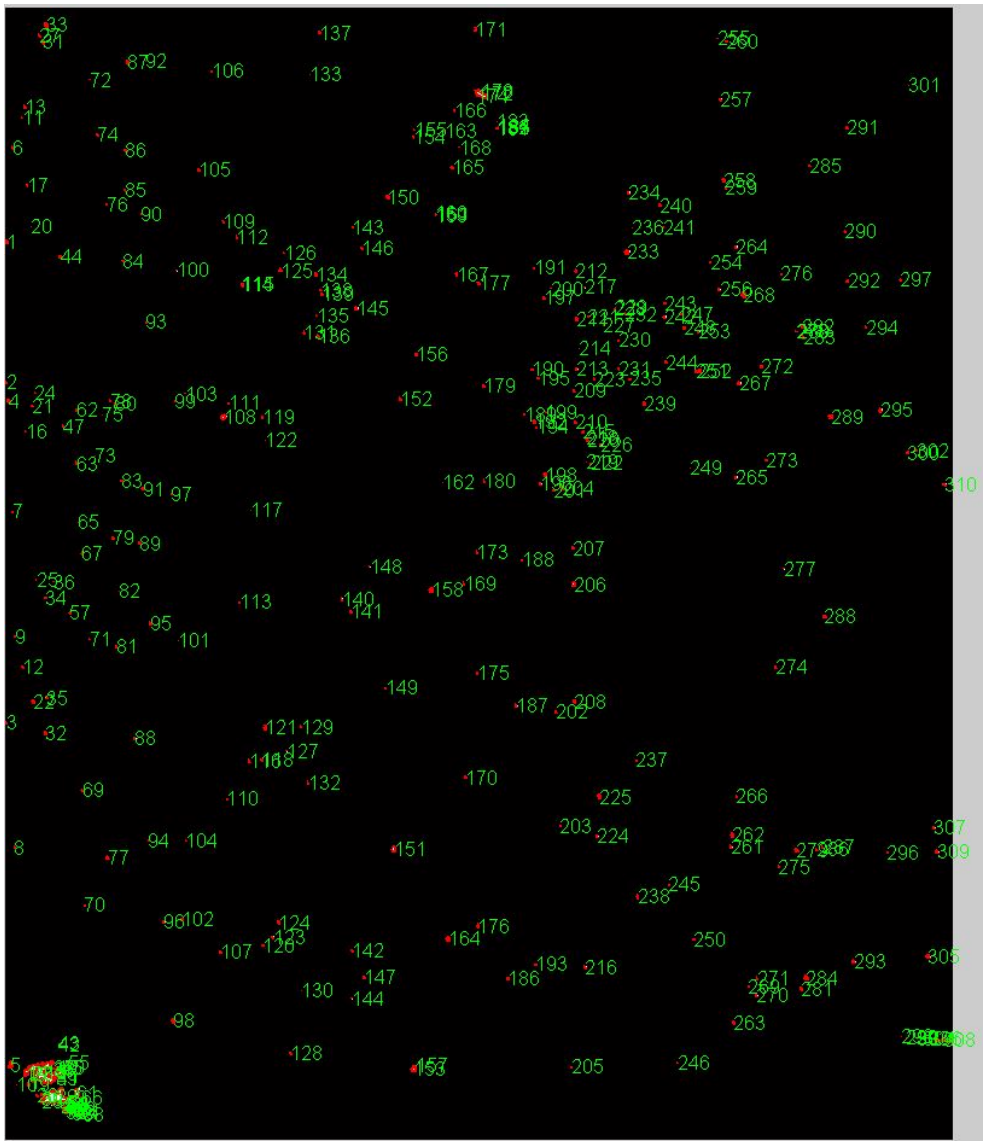
Once I succeeded in displaying the numbers on the image, I compiled a document that identifies each of the different particle sizes, has a corresponding particle image for each, identifies the perimeters of the chosen particles, and lists the number of particles there are for each size. The full chart can be found in the followed document: LIGO-T1400491-v1 under the title of “Images and Numbers of the Different Sizes of Resolved Particles.” I discovered that the reason particles of the smallest diameter have a perimeter of zero according to MATLAB is because they resolve as a single

pixel. As seen in the image below, MATLAB's attempt to outline the particle resulted only in a single red dot in its center of the particle, which shows that MATLAB cannot find the perimeter of a particle only a single pixel in size.



I found that the larger the size, the fewer particles there are of that same size, which could be for several reasons. First of all, there could simply be more small particles than large particles on the optic, but it is also because there is a larger gap between particle sizes when they are smaller than when they are larger; the smallest two sizes are 10.0066881 um and 14.15159403 um with a gap of 4.15 microns between them, while two of the adjacent larger sizes are 93.33617298 and 93.87105514 with only a .5 micron difference. Therefore, the system resolves the particles that are in these large gaps between the smaller sizes to the nearest pixel and places them in the category closest to their size, which inflates the number of pixels of that particular size. While this grouping does not mean the overall number of particles detected is incorrect, it does mean that the smaller particles may not necessarily be the exact size they are measured as, their true size within a couple of microns of the measured particle. The number of particles there are of the same size steadily drops from 229 at 10.0066881 um to 72 at 24.51127986 um to 34 at 42.45478208 um and finally down to single digits around 60 um. Also, the gaps between the sizes go from multiple microns to around one micron at around 33 um and to fractions of microns at around 50 um. Therefore, while the lower sizes should not be completely discounted, it is important to remember that for particles smaller than 33 microns, the particles may, in reality, have a diameter within several microns of the size detected and for particles smaller than 50 um, the particles may have a diameter within a micron of the size detected.

As a final test, I took an image of the optic once half of it had been cleaned using First Contact and cropped it so that I had two separate images, one of the dirty half of the optic and one of the clean half of the optic. I ran the images through the code and found that MATLAB detected 310 particles on the dirty half and 6 on the clean half with diameters of 14.151594 um, 24.511 um, 33.1884 um, 17.332 um, 10.006688 um, and 24.511 um. The processed images can be seen below. We simply painted the first contact on, waited for it to dry, and peeled it off, without any further steps, which accounts for there still being particles on the clean part of the optic. Therefore, the results show that MATLAB can successfully detect the particles on the optic and convey its cleanliness level.



Dirty half of optic

Clean half of optic

Looking Forward:

I have almost completed the objective of the project as I have determined a replicable, optimum camera setup for taking images of the optics and am able to take clear, consistent images of the optic. I have also written a computer code to detect and measure the particles on the images down to 10.0066881 microns with a maximum margin of error of 4 microns. My remaining tasks are adding a command to the code to create a bar graph of the results and writing a final report.