Advanced Measurement Techniques in Fluid Mechanics and Heat Transfer

Prof. Saptarshi Basu
Department of Mechanical Engineering
Indian Institute of Science, Bengaluru

Week – 04
Lecture - 18
Experimental Image Processing in ImageJ

Okay, so as I have already explained, ImageA is an open-source toolbox and we'll be using that for processing our experimental image data. So this toolbox was originally developed for medical image processing by the National Institutes of Health, and it is currently used very widely in the scientific community for simple image processing tasks. So the software can be downloaded directly from the internet. And once we load the software, this is the window that pops up. So here we can see that there are multiple options available, and the file panel here basically helps in opening up the image sequence to import it into the toolbox. So when we store the image sequence from a high-speed camera, let's say we are recording at 1000 FPS and we record the video for one second.

Basically, we have 1,000 images saved in a folder from the high-speed camera. So this entire image folder can be loaded into the imaging. And once we have loaded the image, we can perform multiple operations on it. We can duplicate the imported image stack.

We can perform an image inversion. So this is a very interesting operation. So, usually in a grayscale image, the highest intensity level, 255, is assigned a color of white, and the lowest intensity level, zero, is assigned a color of black. During this image inversion that is reversed. So basically, the highest intensity level, 255, is assigned a color of black, and the lowest intensity level is assigned a color of white; proportionately, the gray values also change.

So this can become helpful under certain conditions, but we won't be exploring this in the current lecture. And then there are operations that we can perform over the entire stack of images, the entire sequence of images that we have imported. So this becomes important and we will be exploring one of the stack operations. Say we might be interested in getting the average of a sequence of images, or we might be interested in getting the standard deviation of a sequence of images. So there we use these stack operations.

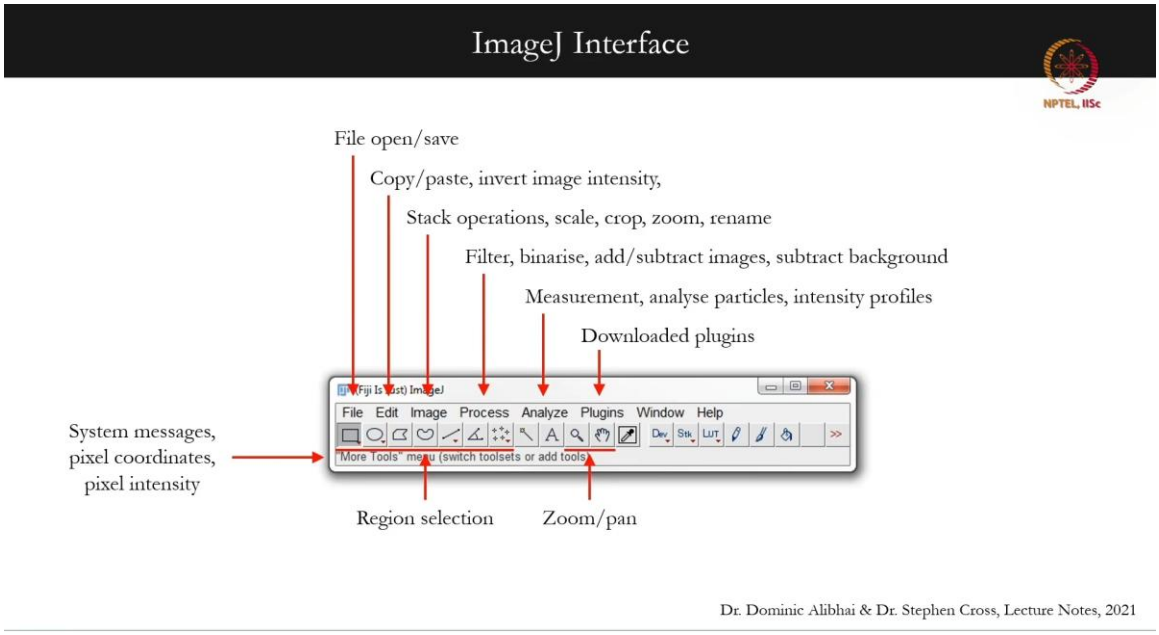
And, of course, we can scale the image that we have loaded. We can crop. We can zoom in on a specific zone. We can rename the stack. All these operations can be performed.

And then there are operations like filtering, which are very essential, and we'll also be exploring a few filtering techniques. So these are essentially noise reduction techniques. So our experimental data is usually very noisy, and we need to filter out unwanted noise from it to obtain useful data. So, we'll be exploring a few filtering methods here. And once we have filtered the image, we need to binarize it.

So the point of binarizing it is to identify the boundary and to identify the object. Of our interest, here in our case we are interested in identifying the flame that is actually moving; so basically, we need to identify the boundary that contains the flame. During that process, we will be binarizing the image, and then there are options to add and subtract images. This is a very important tool, and we'll be using this. This is again very important because our data is noisy in nature, so there is background noise.

So, we need to remove that. So, the technique that is usually used is to take our image and subtract the background image from it. We will be performing that in the subsequent slides. And then there is a panel to analyze the image data. So, this comes after we have identified the object of our interest.

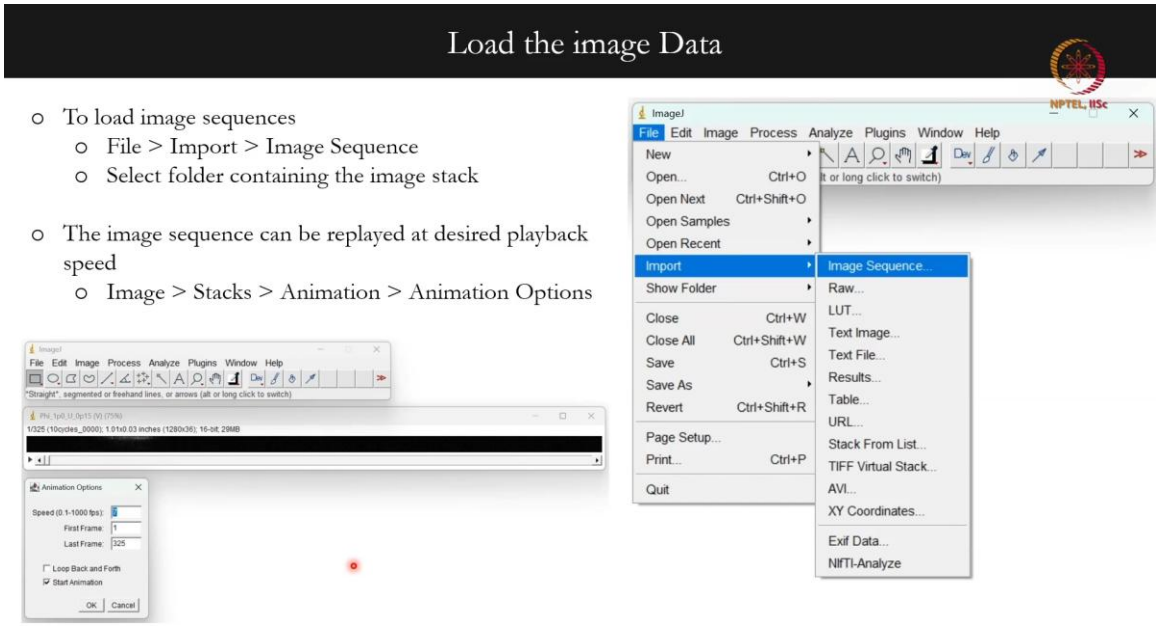
So, once we have isolated the boundary that contains our object of interest, we can use this panel to measure the dimensions of the region of interest, to measure the intensity profile, and functions like that. We will look into a few of these. Additionally, there are plugins and macros that can be downloaded, and they appear in this panel. We won't be using any of these in the current lecture. So, in the bottom panel, we will get information regarding the coordinates of the pixel at which our cursor is present, and it also shows the total number of pixels in the horizontal and vertical directions. Along with that, there is a toolbox that has options to draw a line, draw a rectangle, and draw ellipse polygons. The idea behind this is to draw a region of interest across which we are going to perform further post-crossing operations. Again, we'll be using one of these, and we'll demonstrate this in the further slides. And again, there are options to zoom in, to pan the image, and we'll look into a few of these in the further slides.



Okay, so as I already mentioned, we can load a stack of images or a sequence of images from a folder directly into Imaging.

So, we can use the file panel and, from import, we can select the image sequence. And once we have loaded the image sequence, we can play it at a specific playback rate using the play button here. We can even change the rate at which we play this animated video. So for that, we can go to the image panel under Stacks. So there is animation, and then there are animation options where we can change the playback rate.

So this helps us to view the image sequence as a video in the form of an animation, basically.

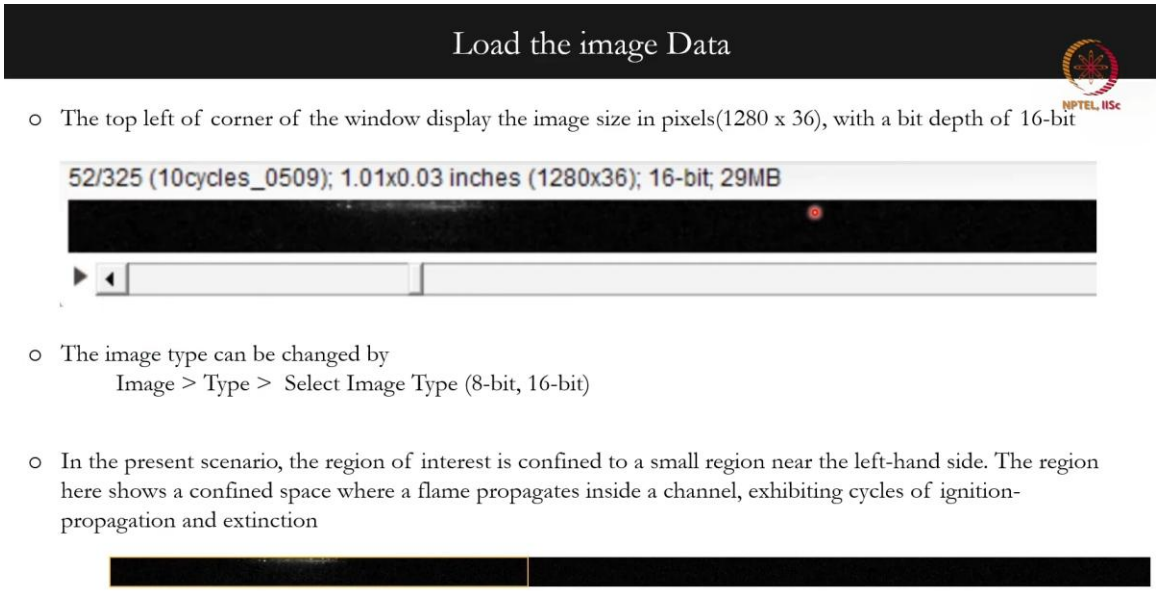


And once we have loaded the image sequence in the panel, we can see that this is the information regarding the image sequence that has been loaded. So here it shows that this stack essentially has 325 images, and the image that is currently being displayed is a 50-second image. This is the name of the image stack that has been loaded. Basically, this was the name of the folder from which this image stack was imported.

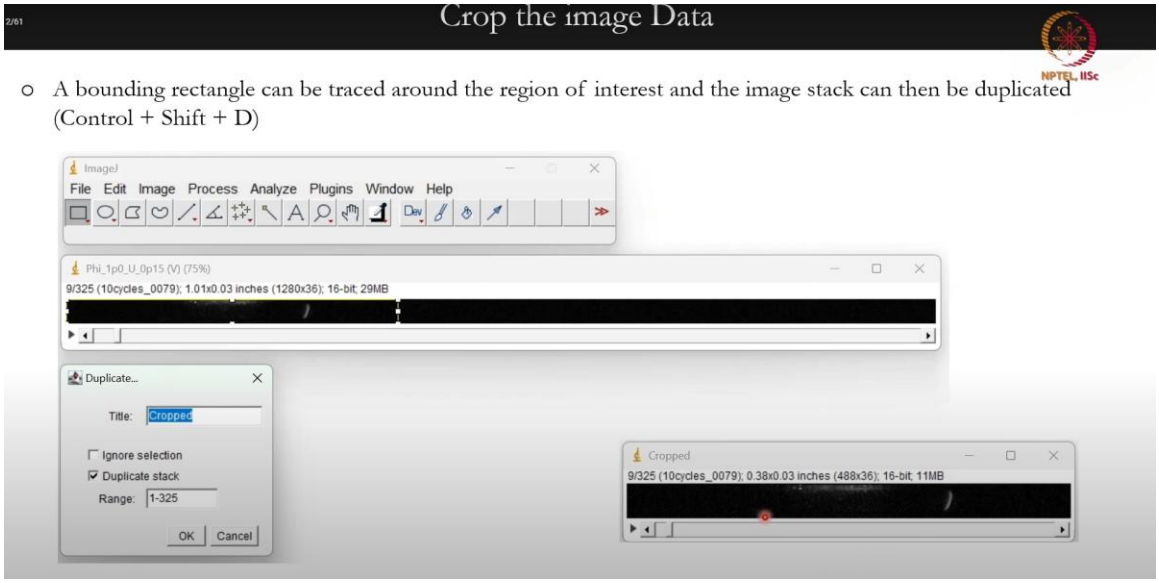
And here this shows the scale in metric units, and this shows the scale in pixel units. So basically, what this means is that 1.01 inches is equal to 1080 pixels in the current case. So this is a scale that can actually be changed, and I'll show in the following slides how to change the scale. Also, we can see that the loaded image set is 16-bit.

It is 16-bit and it is grayscale in nature. Again, this image type can also be changed. We can change the image type to, say, 8-bit as well. But there is an option for that. Under the image panel, we can select a type, and then we can change the image type.

Okay, so now to proceed with further processing of our image data, if we look into the animation of the loaded image sequence, we will see that the flame is actually confined to a very small region in our image, so processing the entire image is of no use. So what we do is We use the tools from the toolbox. We draw a rectangular area of interest in our image where the flame is actually present. Once we have this rectangular box loaded, we can duplicate the stack using this rectangle as the boundaries. And once we do that, we basically create a new stack.



That is confined within a rectangular box. The idea behind this is that we'll only be performing the further post-processing operations on this smaller region because performing them on a larger region takes more computational time. So, we can see that the cropped image sequence we have here is highly noisy in nature.

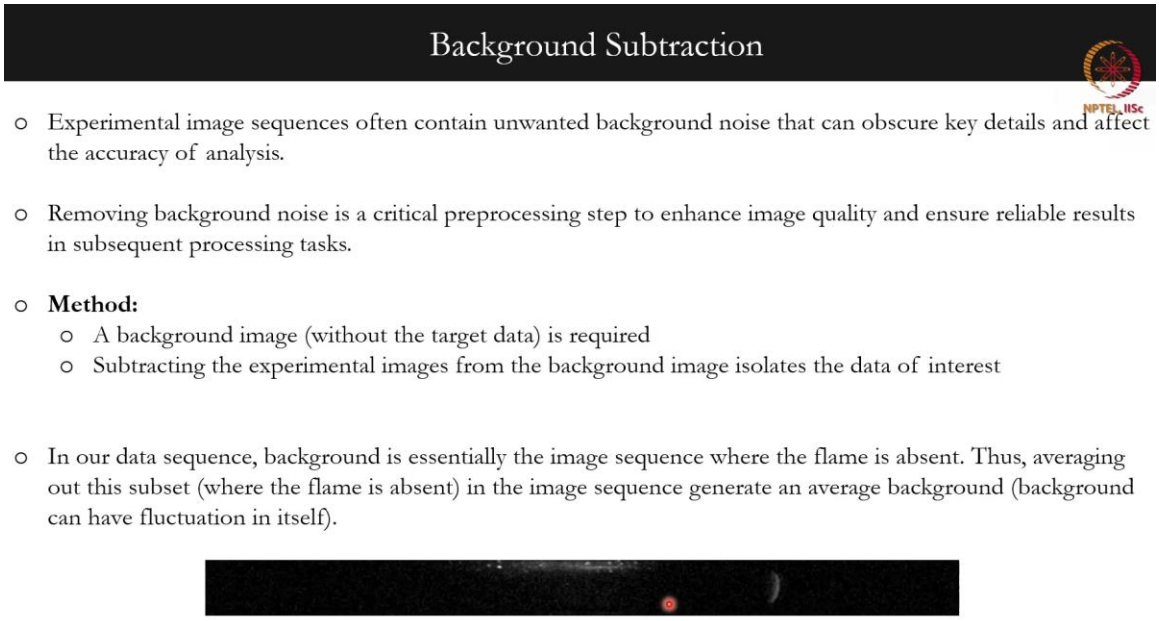


For example, there are pixels here that are always at high intensity, whether the flame is present or not. So this is essentially the background noise, and this needs to be eliminated from our data so that we can process it.

So the most common technique used for background subtraction is to select a background image and then subtract the background intensity from the noisy signal. Uh, so to do that, we first need to make a background image. Usually, in experiments, a

background image is taken so that post-processing can be done, but in our case, we don't need to take a background image, and the background image is already present in this stack of images. We can see here that the flame is present only for a very short sequence of time. For the majority of the time, the flame is not present, and when the flame is not present, it is basically just the background.

So, from this sequence of images, whatever I have loaded, I can basically select a subset where the flame is not present, and that is essentially my background. And once we have that background, we basically say we have selected the sequence of images where the flame is not present. Since this is basically background noise, there are variations between the images within the subset itself.



That is because noise is essentially random by nature. To get an average profile of the background noise, we need to take the average of all the small subsets that we have taken.

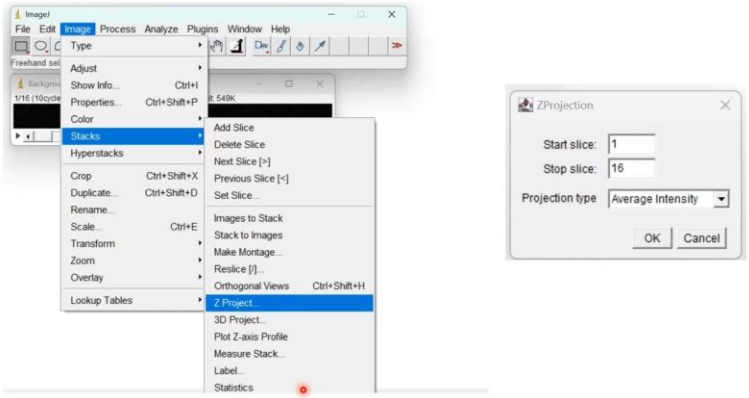
So here, the small subset starts from 1 to 16, and from the image sequence from 1 to 16, basically the flame was not present, which means that it is essentially the background. We are going to average out the intensity and get the average of the intensity across each pixel over this entire stack of images. So that is what this operation does. We go into the image panel, into stacks, and then Z Project. And in the Z project, we select the option of average intensity.

So this basically gives us an average image of all 16 of those images. Basically, it gives the average of those 16 images where the flame is not present. Basically, it's the average of all the background images. So once we have generated this average background image, what we can now do is use that average background image to subtract the noise data from

the original image data. Again, for that, we can perform this subtraction operation.

Background Subtraction

- Generating the background image
  Image > Stacks > Z Project > Projection type (Average Intensity)



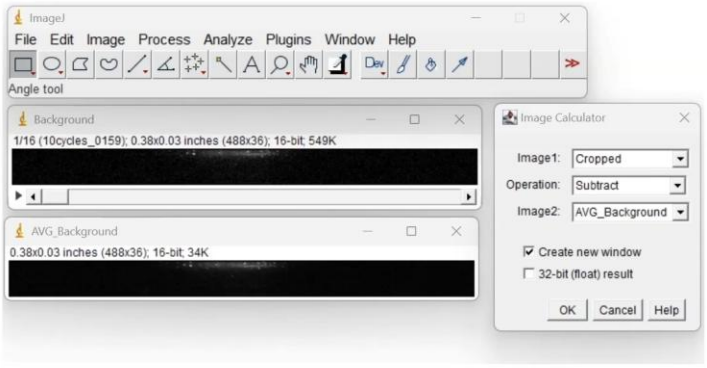
- The **Z-project with average intensity** generates a background image that represents the average intensity (at each pixel) across a sequence of background images.

We can go to Process, and then in Process, we can go to Image Calculator. In the Image Calculator, we can select the operation of Subtract, and now we can perform the subtraction operation. of the noise from the original data. And whatever the resulting image sequence is that we have is shown here. So we can clearly see that the image quality has significantly improved.

The continuous region where there were high-intensity pixels, regardless of whether the flame was present or not in the cropped image, is currently not there. And that is because the background noise has been removed. So this is basically the image from which we have subtracted the background.

Background Subtraction

- Noise filtering
  Process > Image Calculator > Subtract



- Resulting Image Sequence

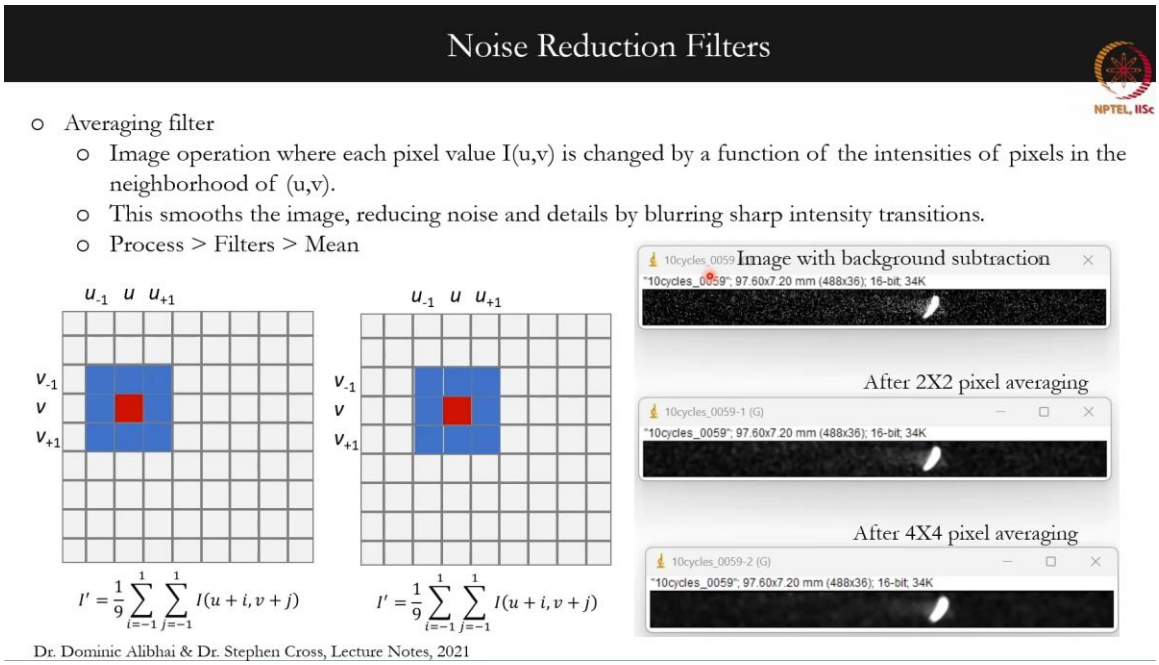
So we can see that even after subtracting the background, there are regions where the pixel intensity randomly shoots up. So, these are isolated regions; they are regions that are away from our main flame boundary, but their pixel intensities randomly shoot up, and these are again a kind of noise that needs to be removed.

It is very essential to remove this noise because, in the further steps, we will see that we are going to isolate the flame boundary based on an intensity threshold. And if there are pixels where the intensity is very high, pixels that are disconnected from the flame boundary where the intensity is very high, the algorithm will map that point as a part of the flame boundary when it is ideally disconnected. So, it is very important to remove these pixels that randomly shoot up, where the pixel intensity is randomly shooting up. So here we are going to use filtering techniques for this. And here, we'll use two filtering techniques.

One is the averaging filter, and the other is a median filter. So in the averaging filter, the function basically creates a bounding box and then changes the pixel intensity values of all pixels in an image. So here I've basically shown a schematic of a 3x3 averaging filter. So basically, what this does is create a bounding box with three pixels along the horizontal direction and three pixels around the vertical direction. And then it centers the bounding box around the pixel where the intensity value needs to be changed.

And then it estimates the average intensity of all the pixels in that bounding box, assigns that particular average value as the new intensity at this location, and then the sliding window moves to the next location, traversing all the pixels in that image, essentially

changing the pixel intensity for all the pixels in that image and generating a new image. So that is what happened here; this was the original image with the background being subtracted, of course. After we do a 2 cross 2 pixel averaging, this is the flame image that we end up with, and after we do a 4 cross 4 pixel averaging, this is the flame boundary that we end up with. So we can see that wherever the random pixels where the intensity was shooting up before filtering, those intensity levels have actually reduced after this averaging filter. Now we'll explore another filter, which is called a median filter.

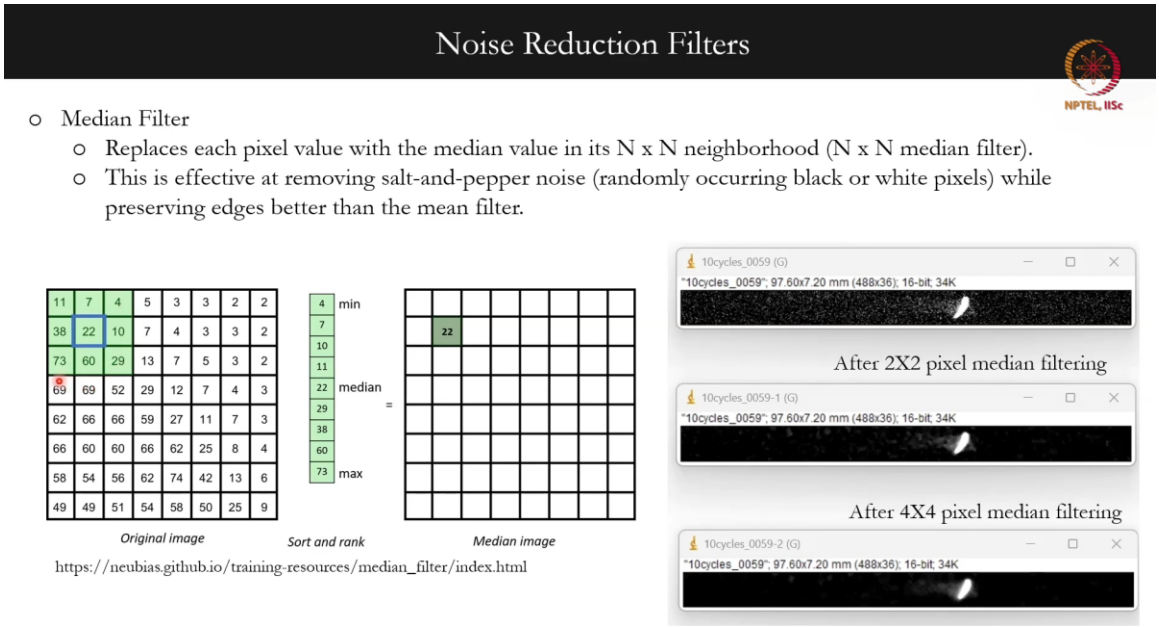


The averaging filter is very similar to the median filter except that instead of taking the average of all the pixel intensities in that bounding box, it takes the median of the pixel intensities in that bounding box. So again, here I have shown a 3 by 3 median filtering. So again, the bounding box has three pixels in the horizontal direction and three pixels in the vertical direction. And the bounding box is centered around the pixel where we are currently changing the intensity. It lists out, sorts basically all the pixel intensity values, identifies the median value, and assigns that obtained median value as the new filtered value of the intensity at that pixel.

And again, this bounding box traverses the entire image, changing the pixel intensities of all pixels. And it generates a new image altogether. Again, we can see here that this is the original image before filtering. And this is the image that we obtained after 2x2 median filtering, 2x2 pixel median filtering. And this is the image that we obtained after 4x4 pixel median filtering.

Again, we can see that the points where the intensities were randomly shooting up have been removed to a great extent, but a very important problem has also been introduced because of this filtering, and that is that the boundary of the flame has been blurred.

Initially, the flame's interface was a little bit sharper, but after this filtering using the median filtering or the mean averaging filter, it has become less distinct. What we have essentially done is blur out the boundary of the flame. However, we will tackle this problem next when we identify the flame boundary using the threshold technique. So, once we have filtered the image, we need to identify the boundary of the flame.



We already saw that when we use filtering techniques such as average filtering or median filtering, we essentially blurred the boundary of the flame. Now, to capture the boundary and get the effective boundary of the flame, we are going to use this thresholding technique. So thresholding essentially converts our grayscale images into a binarized image. So, the technique for creating this is quite simple. So, basically, I decide on a cutoff value; say, here the cutoff value is 124.

So, that means that all the pixels with an intensity value greater than 124 will be assigned a binary value of 1, and all the pixels with an intensity less than this value of 124 will be assigned a binary value of 0. And all these pixels that are identified by the binary value of 1 will be called foreground. The class will be called foreground. The class of pixels identified by the binary value of 0 is called background. So, this is the essential idea behind this thresholding basically to convert a grayscale image into a binary image.

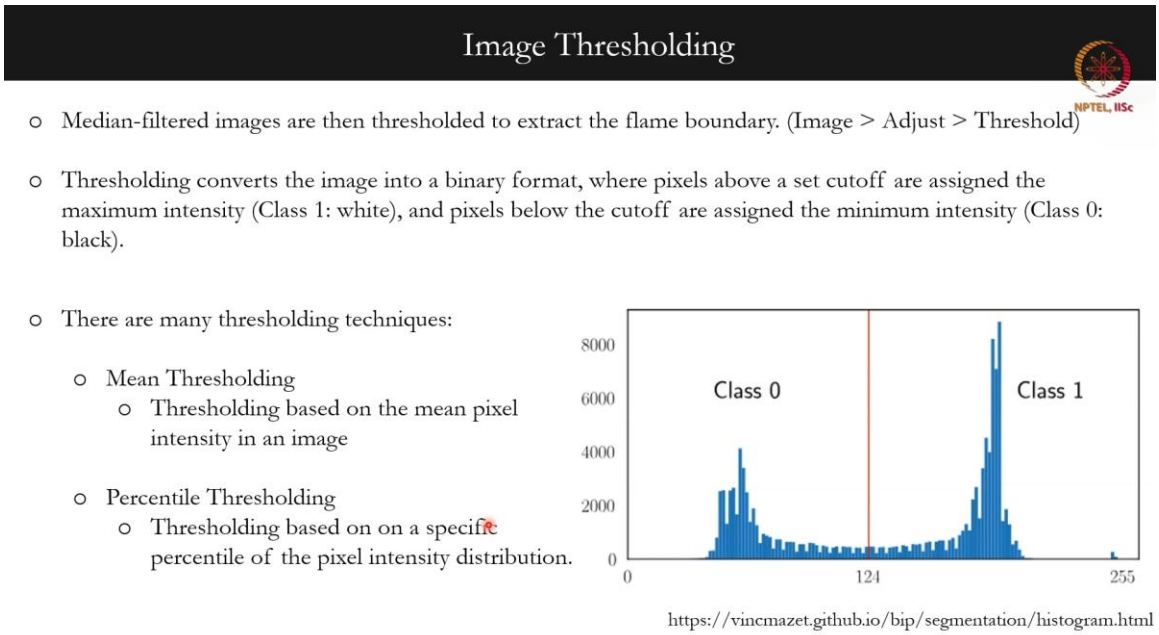
So, uh, this can be done in multiple ways. Here, actually, I have shown a histogram of an image. We can see that, from 0 to 255 intensity levels, what the number of pixels associated with each level is. For example, at an intensity level of 124, we have approximately 500 pixels. This is basically the distribution of pixels across different intensity levels. And once we choose a threshold value, everything on the right side of this threshold will be assigned a binary value of 1, which basically becomes a foreground, and everything on the left-hand side will be assigned a binary value of 0, which is

basically the background.

Now, this cutoff or threshold value can be chosen in multiple ways. So we'll explore three techniques here. So one is a mean threshold. So in mean thresholding, what we do is take the average of all the pixel intensities in an image and then use that mean to separate the foreground from the background. So if the mean value is, say, IK, we say that all the intensities which are greater than IK will be class one, which is basically foreground, and all the intensities which are lower than that mean value, which is again IK, will basically be the background.

There is another thresholding technique that we can use, which is basically the percentile thresholding. Say here again, I'll explain this with an example: if I say that I'm going to do a two percent percentile thresholding, what that means is that... The 2% of the pixels that have the brightest intensity will be assigned to the foreground category, and the remaining 98% of the pixels will be assigned to the background category.

So, only the two brightest pixels will be categorized into the foreground and assigned the binary value of 1. We'll see what the effect of using different thresholding techniques is.

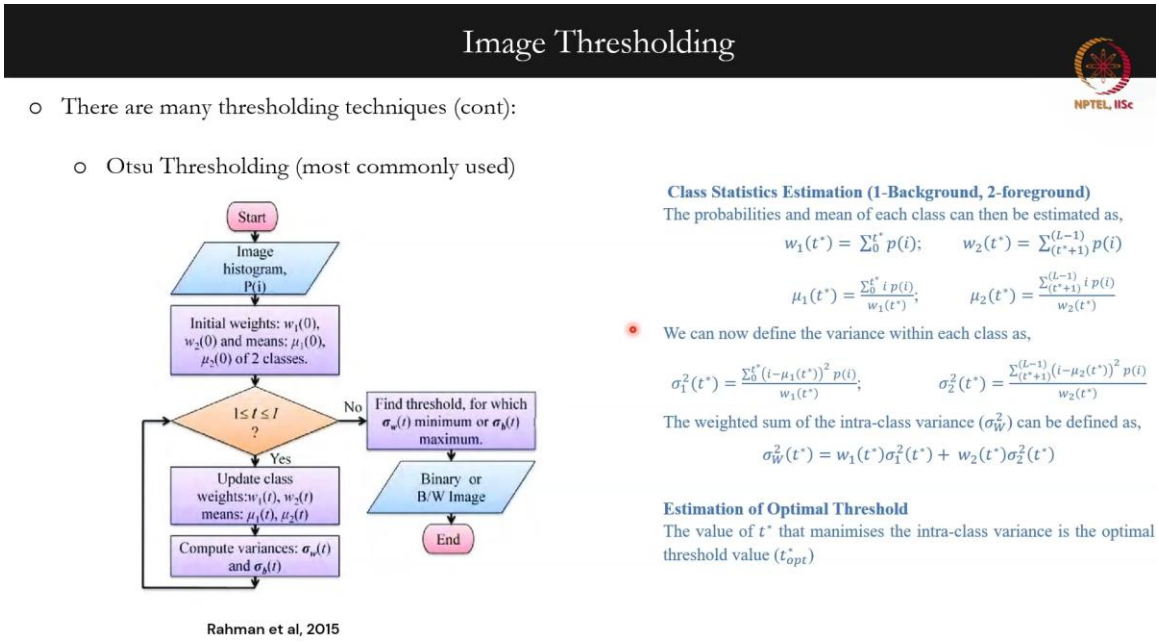


Before that, we'll go into another thresholding technique called the OTSU thresholding. So, this is a very commonly used technique in image processing. So what we have here is that it iteratively estimates this threshold value.

So the idea behind this is that we start off with an initial guess threshold value. And then we categorize the image into the foreground and background. And once we have categorized that, we can calculate the class statistics for each of the foreground and

background classes. So we can estimate what the probability of occurrence is for each class.

We can estimate what the mean is associated with each class. We can even estimate the standard deviation and variance within each class. After that, we define a parameter known as the weighted intraclass variance, and the threshold is iteratively chosen such that this value, which is basically the intraclass variance, is minimized. Mathematically, it is possible to show that minimizing this value of intraclass variance maximizes the variance between the two classes, which are background and foreground. So basically, in an iterative manner, the OTSO thresholding algorithm chooses an optimum value of threshold where the variance between the background and the foreground is maximized and the variance within each category of the background and foreground is minimized.

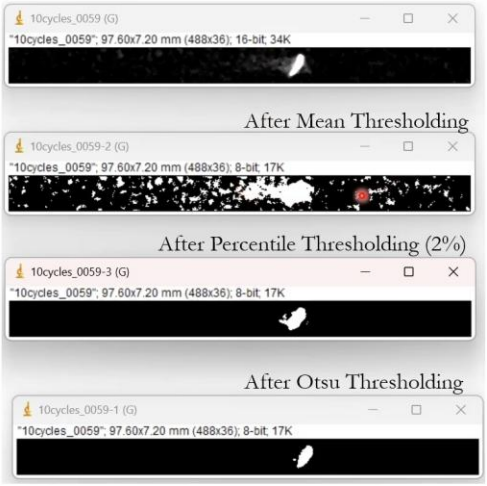


So this is basically the sequence of the images that we get after thresholding; this is the image before thresholding, and this is after mean thresholding.

We see that mean thresholding does not do a good job, and then we can do better. Here, I have shown percentile thresholding with 2%; basically, the 2% brightest pixels are assigned a value of 1, and this is for the OTSU thresholding. We see that the OTSU thresholding technique does a pretty good job at capturing the flame boundary. Now that we have actually estimated the boundary of the flame, we have identified our object of interest. Now we can use the analyze toolbox to actually capture the parameters such as the flame dimensions and other associated factors.

Image Thresholding

- There are many thresholding techniques (cont):



So once we have the binarized image, this is the image sequence that we end up with. So to sum up everything, we started off with a noisy image, then we did background subtraction, followed by noise filtering, and finally, we did binarization. And this is the image sequence that we finally ended up with. So here, the pixels only have two values.

They can either be of intensity 0 or of intensity 1. They are binary images. Okay, and now we can use these binarized images to basically isolate the boundary of the flame. Before that, before going into this, I'll show you how to set the scale for the problem. So for that, here basically setting the scale means how to change the spatial resolution or how to change the metric that is represented by a pixel dimension. So, here if we go to analyze and set the scale, we can change what each pixel represents.

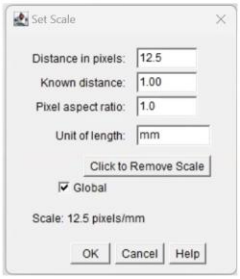
So, basically, here I am saying that 12.5 pixels represent 1 mm in my problem. So this is actually constrained by the high-speed camera that is used to capture the imaging system that is basically used to capture this phenomenon. A calibration is usually taken during the experiments to estimate what the pixels per mm are. And these values are directly imported into this when we set the scale. So once we have actually set the scale, the point of this is that we can now obtain all the parameters of our interest.

Feature Extraction

- Image sequence after back subtraction + noise reduction + thresholding

- Before we estimate the temporal variation of the flame boundary, we need to set the spatial scale (spatial resolution at which the image sequence was captured).

  Analyze > Set Scale



- Once we have set the spatial scale of the image, we can now measure the parameters in the image (distance between 2 points, area, etc) in metric units

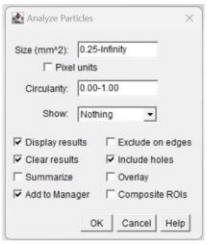
For example, the dimensions of the flame and all the associated parameters in metric units, such as millimeters or meters, depending on which we are interested. Now we can use the analysis toolbox to actually capture the boundary of the flame. To find out how to draw the bounding box around the flame to estimate how the flame is moving in the sequence of images, all these operations can be performed. So, in the analyze toolbox, there is an option to analyze particles, and we can set the thresholds, the lower and upper limits for this. And once we have said that, basically it tracks regions that have intensity and area between these lower and upper limits.

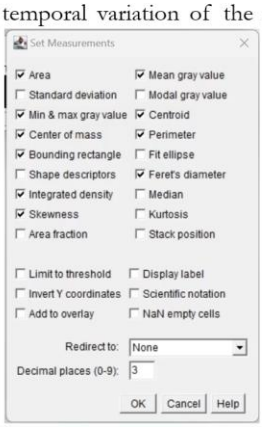
So here it's set from 0.25 to infinity, and anything below 0.25 mm square is discarded, while anything above 0.25 up to infinity is tracked down. And since here we only have one continuous region of pixels that is identified as foreground, it is only going to track the flame boundary. And we can even set, during this analysis process, what parameters we are interested in tracking.

We might be interested in tracking the area of the flame's boundary. We might be interested in the center of mass or the centroid, or we might be interested in the dimensions of the bounding box that encloses this flame boundary. So, all these parameters can be tracked across the sequence of images to obtain a temporal variation of these parameters. Quantities like integrated density do not seem to make any sense in a binarized image. However, I will show how this can be used to make sense, for example, to obtain the chemiluminescence data in the following slides. So once we are done with this, we can run this analysis particle for the entire sequence of images, and we can load all the boundary data, the boundaries of the flame images that are identified from the binary images, into this ROI manager.

Feature Extraction

- The function 'Analyze Particles' can be used to estimate the boundary of the flame (pixel white in color in the binarized image)
  Analyze > Analyze Particles
- The operation when performed over the entire image sequence, give us the temporal variation of the flame boundary





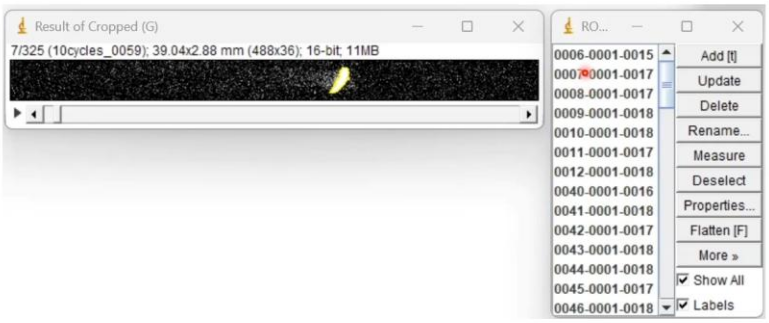
- Features to be extracted can be set with Analyze > Set Measurements
  - Example: Area, Centroid, Perimeter, Integrated density, etc.

So this ROI manager contains information across the entire image sequence showing where the flame boundary is. Now, the very important thing to do here is that once this flame boundary is identified, we need to go back to the original image where binarization has not yet been performed. Because when binarization is performed, all the values within the flame boundary are automatically mapped to a value of 1. So, estimating the intensity of the flame based on the binarized image does not make any sense. Whereas if we estimate the intensity of the flame in an original image where binarization has not been done, there will be a gradual variation in the intensity across the flame along the spatial direction.

So, integrating the pixels and the intensity of the pixels across the entire flame boundary would be representative of parameters such as chemical luminescence, and that is what we are interested in. So, that is what we are going to do next. So, once we have estimated the boundaries of the flame and captured these boundaries in the ROI manager, we go back to the original image, which has not been binarized yet, and we are going to re-perform this entire measurement, whatever we did on this image sequence, which has not been binarized yet.

Feature Extraction

- The analysis results are saved into an ROI manager. Now these boundaries can be traced back into the original image as well



- Estimations such as integrated density (which is a measure of the overall Chemilumnscence intensity – a crucial parameter in combustion studies) can be reperformed with the estimated boundary (stored in the ROI manager) on the noise filtered image sequence (image sequence before binarizing).

So, now, once we do this operation, we are basically in the results tab, where we end up with a sequence of results across the image sequence. So each row basically represents results corresponding to a specific image sequence, and it has been performed over all the 325 images that were present in the stacks that were actually loaded up.

So once we have this data, we can directly export it into Excel format or CSV format. And once we have that, we can plot them.

Feature Extraction

- Results are summarized in the results tab and can be exported to .csv format with File > Save as

Results

File Edit Font Results

	Area	Mean	Min	Max	X	Y	XM	YM	Perim.	BX	BY	Width	Height	Feret	IntDen	Skew	RawIntDen	FeretX	FeretY
1	0.403	7046.079	3079	11103	21.613	1.306	21.623	1.311	3.874	21.040	0.640	0.960	1.200	1.331	2840.979	0.326	443903	264	23
2	1.133	16210.881	4146	30577	24.434	1.382	24.451	1.411	5.645	23.600	0.480	1.440	1.840	2.241	18363.686	0.141	2869326	295	29
3	0.902	15354.617	4946	29854	26.929	1.408	26.947	1.433	5.626	26.320	0.480	1.120	1.840	2.006	13856.006	0.303	2165001	329	29
4	0.698	12409.459	4946	23981	28.348	1.429	28.366	1.444	4.588	27.920	0.560	0.800	1.760	1.847	8656.838	0.329	1352631	349	29
5	0.576	9968.378	4655	18194	29.250	1.408	29.265	1.425	4.448	28.880	0.640	0.720	1.600	1.670	5741.786	0.451	897154	361	28
6	0.429	8832.925	3626	14756	29.849	1.402	29.868	1.414	3.863	29.600	0.640	0.480	1.520	1.539	3787.558	0.313	591806	370	27
7	0.333	7967.250	3669	12330	30.225	1.448	30.236	1.454	3.609	30.000	0.720	0.400	1.440	1.449	2651.501	0.009	414297	375	27
8	0.627	8192.276	5006	13701	21.785	1.414	21.819	1.386	5.041	20.960	0.560	1.360	1.520	1.888	5138.195	0.413	802843.000	262	26
9	1.050	16930.098	4910	30198	24.765	1.417	24.782	1.429	5.306	23.920	0.480	1.440	1.920	2.224	17769.830	0.089	2776536.000	299	29
10	0.877	15024.978	3413	28640	27.140	1.418	27.159	1.435	5.400	26.480	0.480	1.200	1.840	2.006	13173.901	0.236	2058422.000	331	29
11	0.672	12244.457	3331	22365	28.463	1.430	28.497	1.438	4.721	28.000	0.560	0.800	1.760	1.847	8228.275	0.278	1285668.000	350	29
12	0.576	9751.744	3754	16906	29.320	1.405	29.338	1.432	4.428	28.960	0.640	0.640	1.600	1.670	5617.005	0.295	877657.000	362	28
13	0.410	8842.078	4946	14132	29.900	1.393	29.916	1.409	4.175	29.600	0.640	0.560	1.520	1.553	3621.715	0.046	565893.000	370	27
14	0.288	7679.467	4574	11780	30.280	1.453	30.292	1.453	3.316	30.080	0.800	0.400	1.360	1.369	2211.686	0.319	345576.000	376	27
15	1.037	9216.105	3120	16165	22.080	1.367	22.117	1.363	6.332	21.120	0.480	1.600	1.760	2.274	9555.258	0.363	1493009.000	264	28
16	1.101	16303.680	4694	30574	25.072	1.411	25.099	1.431	6.087	24.320	0.480	1.280	1.840	2.197	17947.091	0.127	2804233.000	304	29
17	0.870	14464.588	4910	29132	27.330	1.409	27.349	1.432	5.427	26.800	0.480	1.120	1.840	1.968	12589.978	0.440	1967184.000	335	29
18	0.672	11744.314	4902	22207	28.609	1.418	28.631	1.431	5.447	28.240	0.560	0.720	1.760	1.824	7892.179	0.339	1233153.000	353	29
19	0.480	10115.827	4993	16156	29.389	1.435	29.408	1.444	4.089	29.120	0.640	0.560	1.600	1.632	4855.597	0.179	758687.000	364	28
20	0.429	8424.701	2871	14629	29.941	1.429	29.958	1.430	4.382	29.600	0.720	0.560	1.440	1.475	3612.512	0.379	564455.000	370	27
21	0.275	7750.558	4873	10509	30.321	1.407	30.328	1.415	3.118	30.080	0.720	0.400	1.360	1.381	2132.954	-0.119	333274.000	376	26
22	0.256	6922.825	3207	11062	30.542	1.434	30.551	1.431	2.930	30.320	0.800	0.400	1.200	1.224	1772.243	0.102	276913.000	379	10
23	1.146	10290.402	3034	18523	22.344	1.407	22.404	1.384	5.872	21.360	0.480	1.600	1.840	2.326	11788.685	0.258	1841982.000	267	29
24	1.018	17044.811	4862	29968	25.375	1.412	25.392	1.427	5.513	24.640	0.480	1.280	1.840	2.130	17344.800	0.070	2710125.000	308	29
25	0.813	14381.244	4861	28941	27.475	1.411	27.502	1.429	4.928	26.880	0.560	1.040	1.760	1.968	11689.075	0.359	1826418.000	336	29

So here I'll show a plot of how the chemiluminescence was estimated based on the integrated density again. Since we are performing this operation again on images that are not binarized and on the 8-bit images or 16-bit images, integrating all the pixel intensities

inside the flame boundary is representative of the total heat release rate or the chemiluminescence intensity of the flame. So this is how the chemiluminescence intensity profile looks here. And here I've basically plotted how the centroid of the flame moves from ignition to extension.

So we can see that as the flame ignites at a specific location, the chemiluminescence intensity rises, reaches a peak, and then gradually starts decaying. During this entire process, the flame upon ignition starts moving upstream along the positive x-axis. It then propagates, and at a particular point, it extinguishes. So here, what we have done is that from the image, we have been able to quantitatively track the temporal variation of a few parameters. There are many other parameters that can actually be plotted; we can estimate, based on this data, how the flame shape is varying during this entire propagation phase, and many other parameters can be plotted to get their temporal variations.

And all these can be used to make sense of the data, sense of the physical phenomena that are actually happening. What we finally see here is that, from all the images, we are going to extract time series data again. And this is very similar to the data that we get from a pressure sensor or any sensor-based devices as well. So in the next portion of this lecture, we'll be looking into how to make sense of these time series data.

How to represent them in different coordinate systems so that analysis becomes easier. So, that is what we will be looking into next.