Special Topics (II) Final Report: Post Processing of PSV Images with Gradient Illumination

國立臺灣科技大學

NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Name	: Ivan Nagara Putra Wahyudi (林月王)
Student ID No	: B10735020
Major	: IATP Mechanical Engineering Field
Advisor	: Wei-Hsin Tien

Abstract

The Particle Streak Velocimetry (PSV) method is one method to describe and calculate the particle's instantaneous velocity properties in a velocity field. PSV uses the camera's streaking image and calculates the trailing streak of the moving particle's length corresponding to its exposure time to calculate its instantaneous velocity properties. PSV can be done without the requirements of expensive hardware; usually, other methods require an expensive camera as. PSV opens an opportunity window to those who may need to do velocimetry experiments but do not have any sufficient/adequate requirements of camera specifications.

The algorithm which will be explained in this paper is conducted by using a MATLAB language-based program. The algorithm has been well-developed by Mumtaz Hussain Qureshi and Wei-Hsin Tien. The algorithm calculates the polynomial equations for the streak's movement path, its instantaneous velocity at given points, particle diameter, and the particle squared norm of the residual value (Resnorm) which indicates the error of the corresponding results. The algorithm uses the least-square curve fitting method, which calculates the best fit that will be as close as possible to the real path's trajectory. The least-square curve fitting method will result in a 7^{th}-order polynomial function of time.

With the reliable results of the least-square curve fitting algorithm, the final thing to figure out if the result is right, besides judging from how small the resnorm value is to make sure that the velocity direction is right. With the earlier algorithm which does not involve the correctness of the velocity direction (positive-wards or negative-wards), an additional research method of sample collection and additional algorithm features are needed. This paper will discuss the additional sample collection method, also the new algorithm that is designed to process the newly collected samples accordingly.

Introduction

The Particle Streak Velocimetry technique has been used as a method to solve the velocity properties of particles in a particular velocity field. This method requires a streak image, which is obtained by using a camera with a fixed exposure time to capture the moving particle, to introduce the streak of the particle's movement path. In this specific experiment, the MATLAB programming language is used as a tool to input the calculation and algorithm's logic. This program is mostly written and designed by Mumtaz Hussain Qureshi.

The first set of algorithms and programs was designed to see if the curve-fitting technique/logic is running well because the curve-fitting algorithm is the algorithm being utilized to calculate and track the particle's instantaneous velocity and the streak's path itself. The curve-fitting technique detects the curvature equation shown by the streak's curve. By using the combination of the least square curve fitting technique with the Gaussian intensity distribution, we may then acquire the instantaneous velocities (start, middle, and endpoint), particle diameter, and the squared norm of the residual, etc., values of the streaking particle.

By the end of the first algorithm development, the results are quite promising since the velocity vectors shown on top of the original image are well aligned and seemed to make sense in the direction of their movement (An additional and more scientific check may be done by checking for the streak’s squared norm of the residual (*Resnorm*) value). Even though the velocity vector direction makes any sense, the true directions are unjustified. From the image, the streaks are just a long streaking image without any clues of where the particle's starting point, nor the middle and end point. So, a new idea was developed to identify the starting, middle, and endpoints.

The additional technique to identify these three important points to justify or make sure the velocity vector directions are right is to implement the light intensity variation during the image-taking process. The light intensity is a function of time, so the streak’s intensity may indicate which stage the streak is at. Finally, some additional hardware requirements are added to achieve the hypothesized light intensity variation experiment objective.

Motivation

This research is conducted to continue the progress of earlier research, which is also based on the MATLAB Particle Streak Velocimetry algorithm which has been built and designed by Mumtaz Hussain Qureshi. Earlier research has shown positive and good indications of results output; thus, this research is meant to fix and cover the parts where the earlier research did not manage to cover, also to add some additional features so that the results may be justified as a correct answer.

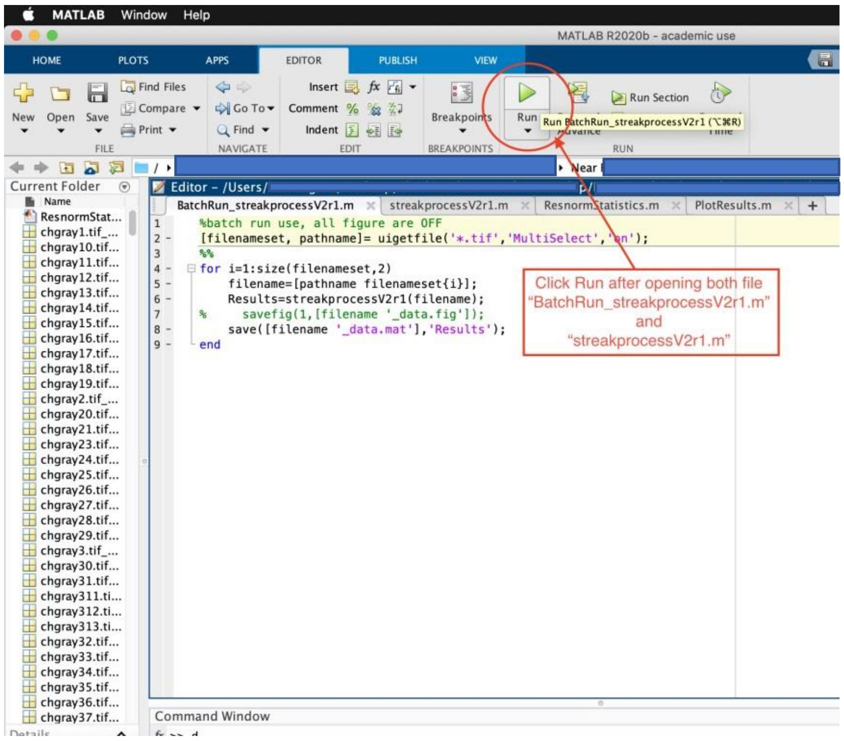
Description of Research Work

Chapter I: Development of the Non-Gradient Illuminated "streakprocessV2r1.m" MATLAB Algorithm

The earlier MATLAB algorithm for solving the Particle Streak Velocimetry problems, which was designed by Mumtaz Hussain Qureshi has proven to show good and near-correct results despite the correctness of their true path direction. The directions can be seen as if the vectors are well-aligned with the true streak particle's image, with a good velocity tangent line prediction. This shows that the algorithm itself is a success in its calculations, least-square curve fitting techniques, and implementation of the Gaussian intensity distribution formula as well.

In this research work, the main goal is to develop the technique where the researcher may obtain information on the streak's beginning, middle, and ending points. This is crucial for figuring out the velocity's direction, so that the researcher may find the flow field well. An additional technique that has been proposed is to use the light intensity variation with respect to time. With this hypothesis in hand, the streak's original direction may be defined by how bright or how dim the light intensity is at the streak's corresponding path location.

One new feature that is introduced in the new "streakprocessV2r1.m" program is that now, the results are shown in one structure called "Results". Inside the structure, there are smaller variables that have different data (**I.e.,** Results.I3_stats1, Results.Resnorm_List, etc.). Secondly, now there is a new algorithm to call the "streakprocessV2r1.m" function, which is called "BatchRun_streakprocessV2r1.m". When running the Batch Run program, the streakprocessV2r1.m program must be opened in another tab, because the Batch Run program invokes the streakprocessV2r1 function. The Batch Run function is used to input multiple images at once so the loaded images are automatically queued to be processed by the "streakprocessV2r1" algorithm. The process of running the "BatchRun_streakprocessV2r1.m" will be by the figures shown below.


MATLAB R2020b - academic use
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Run
Run BatchRun_streakprocessV2r1 (⌥⌘R)
Run Section
BatchRun_streakprocessV2r1.m
streakprocessV2r1.m
ResnormStatistics.m
PlotResults.m
Current Folder
Command Window
Click Run after opening both file “BatchRun_streakprocessV2r1.m” and “streakprocessV2r1.m”

```
%batch run use, all figure are OFF
[filenameset, pathname]= uigetfile('*.tif','MultiSelect','on');
%%
for i=1:size(filenameset,2)
    filename=[pathname filenameset{i}];
    Results=streakprocessV2r1(filename);
%     savefig(1,[filename '_data.fig']);
    save([filename '_data.mat'],'Results');
end
```

Figure 1.A. Starting the “BatchRun_streakprocessV2r1.m” algorithm.

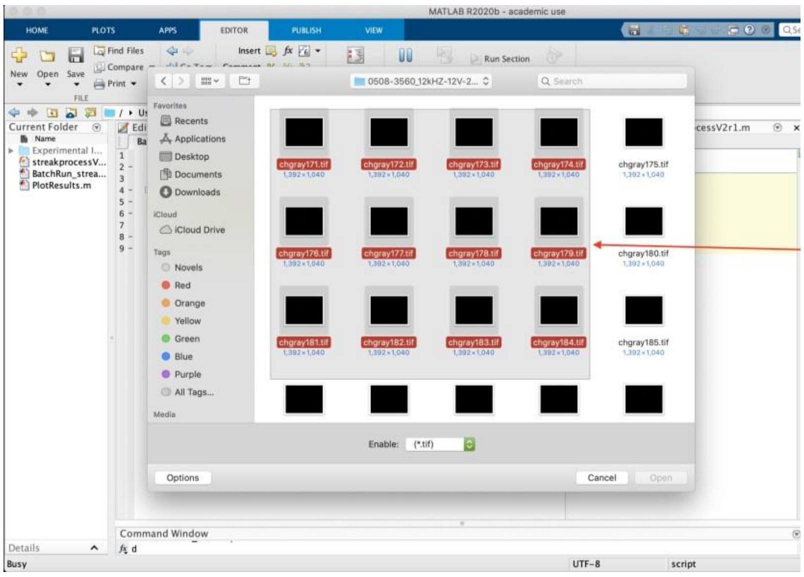


Figure 1.B. Selecting the photos for processing.

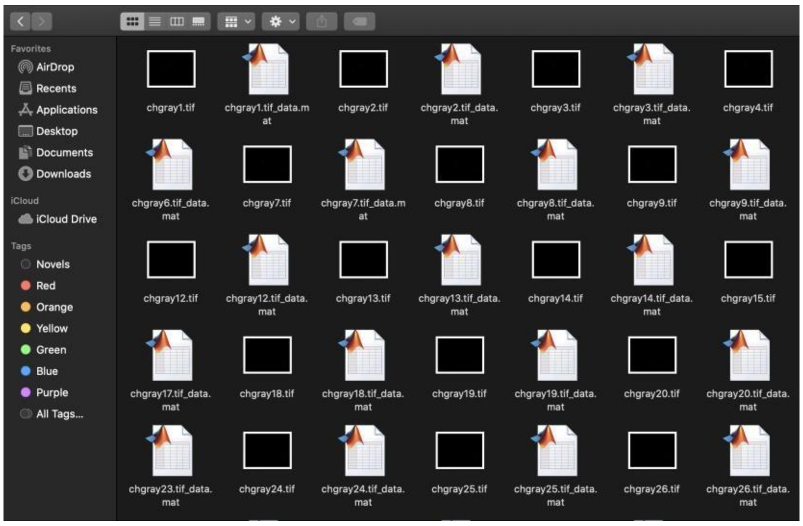


Figure 1.C. The data results of the algorithm.

As we can see from the three figures above, firstly we can run the algorithm by clicking the run button while opening the algorithm's code tab, followed by the selection of the images that are going to be processed by the algorithm. Finally, after the algorithm finished processing the images, the results file with the format of ".mat" will be saved directly to the original image's directory with the same name as the image, with the addition of "_data.mat" in its name.

Third, there is a program called "PlotResults.m" which is used to plot the velocity vectors above the original image file, from the obtained data results of the "streakprocessv2r1.m" algorithm. The "PlotResults.m" algorithm is now available to plot the first, middle, and endpoints well following the streak's spine. The earlier "PlotResults.m" program only used the bounding box top left corner as a reference and it can only plot the starting point's velocity vector in that top left corner. Plotting the middle and endpoint locations is not recommended because they might not be found precisely above the streak's spine.

Finally, a program called "ResnormStatistics.m" was designed to calculate the resnorm population distribution based on their total area in pixel units, because the resnorm values are relative. There is no exact answer to which norm value shows the right calculation of the least square curve-fitting technique. The point is that the lower the resnorm value, the smaller the error is. With the help of this algorithm, it is possible to see the mean, maximum, minimum, and average of the resnorm values. The population group has been divided into six groups, which are Group 1 (100 pixels and above), Group 2 (80-99 Pixels), Group 3 (60-79 Pixels), Group 4 (40-59 pixels), Group 5 (20-39 Pixels), Group 6 (12-19 Pixels). The reason Group 6 only has a minimum area of 12 pixels will be discussed below.

The area threshold chosen minimum area is 12 pixels. The area threshold can be adjusted manually in the “StreakProcessV2r1.m” program in line 66 “thr_area = 12;”. But in this case, the minimum area of 12 pixels was chosen to minimize the risks of noise images and to speed the algorithm up so it can finish processing the whole image faster. The lower the area threshold, the bigger the chance of the noise particle exposure to the whole data.

After some manual reviewing of the streak particles, it was shown that the 8 pixels’ group particles consisted of a lot of background noise particles. It was also shown that starting from the streaks with 10 pixels’ area, there are a lot of background noises included in the data calculation even though the resnorm value is exceedingly small. So, it is concluded that to optimize the algorithm’s speed in finishing the whole iterations and remove the additional risk of including the exposure of additional noise particles, the best area threshold for this experiment is 12 pixels.

Besides the exceedingly small particle which is excluded to reduce the risk of the noise particle addition, there are cases where even particles with a pixel area of above 100 are also noise particles. The difference is that it is better to exclude the smaller particles because usually, they have a bigger population than the large area particles. The error that will be eliminated is much greater when we remove the smaller particles, and it is not effective to remove larger particles because of the chance of them being the noise particles and their population may just be too small.

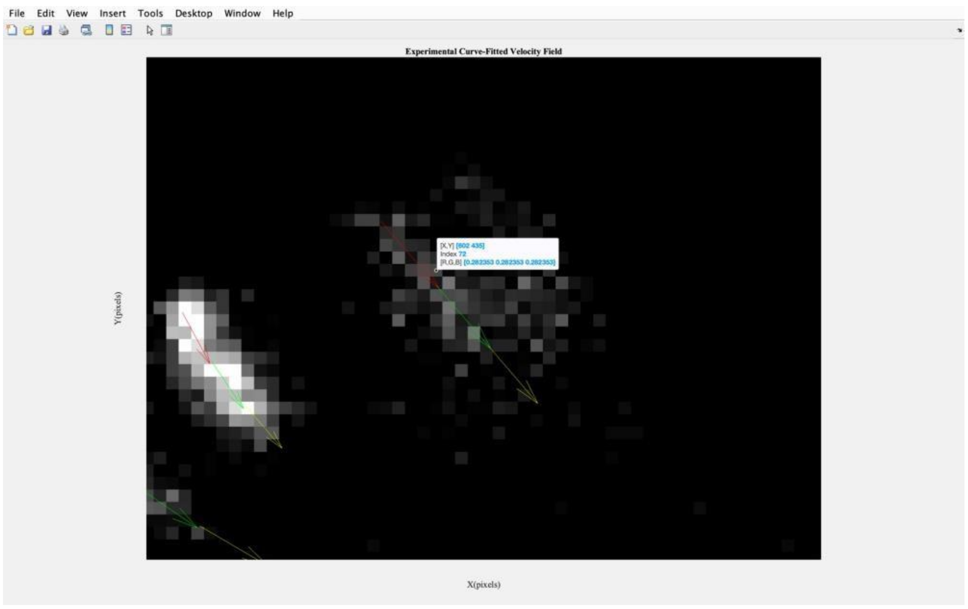

File Edit View Insert Tools Desktop Window Help
Experimental Curve-Fitted Velocity Field
[X,Y] [602 435]
Index 72
[R,G,B] [0.282353 0.282353 0.282353]
Y(pixels)
X(pixels)

Figure 2.A. Class 1 noise particle.

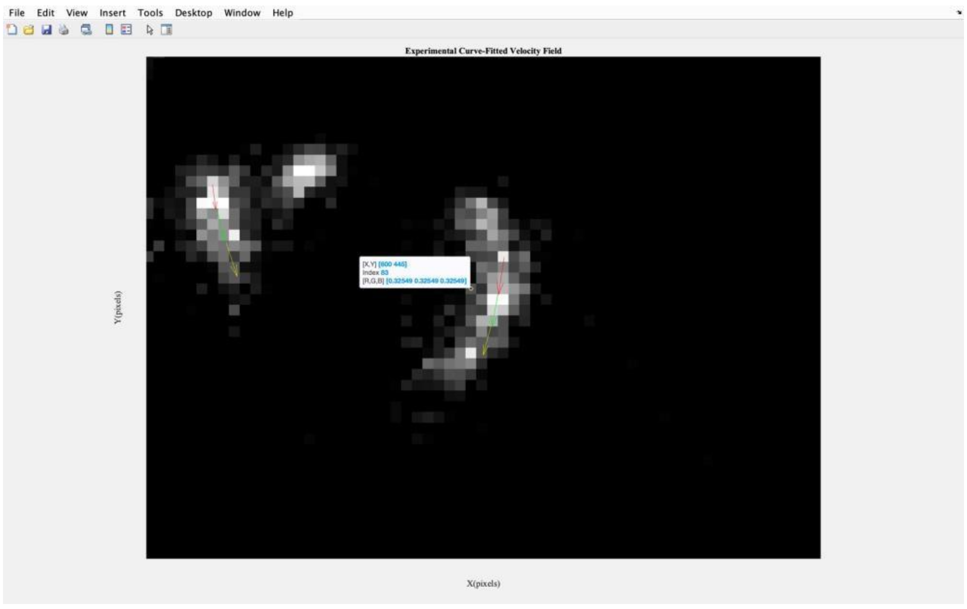
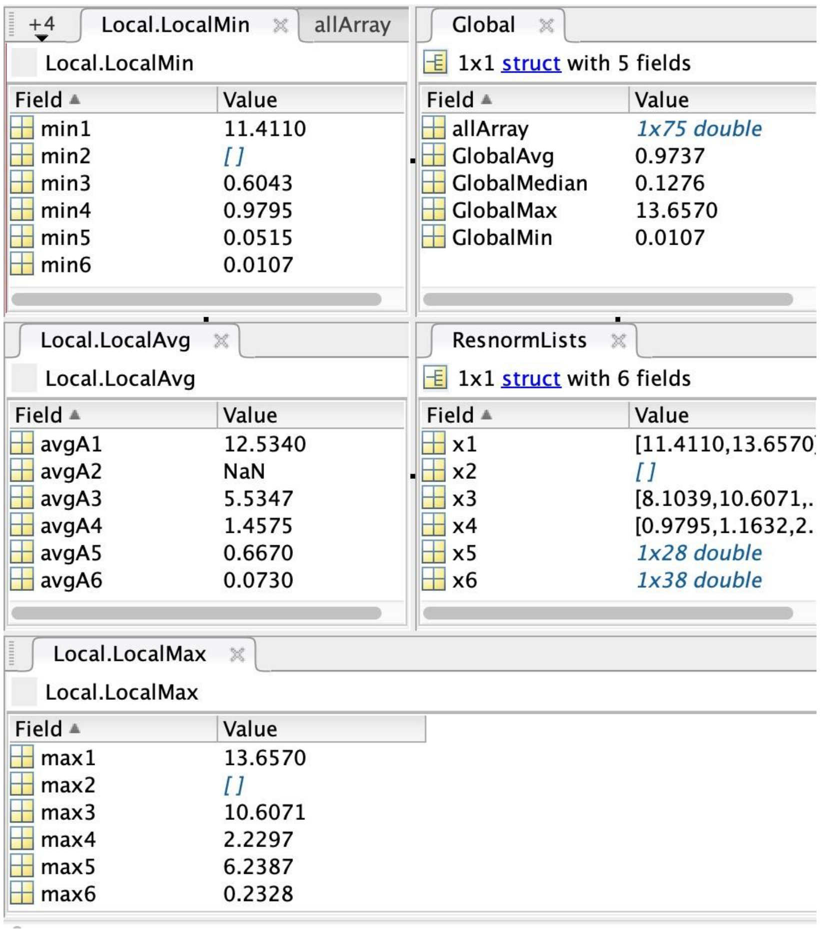

File Edit View Insert Tools Desktop Window Help
Experimental Curve-Fitted Velocity Field
[X,Y] [600 445]
Index 83
[R,G,B] [0.32549 0.32549 0.32549]
Y(pixels)
X(pixels)

Figure 2.B. Class 1 streak particle.

In the 2 figures above, we can see that both (**Figure 2.A.** and **Figure 2.B.**) are classified as Class 1 streaks, which means that they have a pixel area above 100 pixels. But then we can see the difference between them both. **Figure 2.A.** had an abstract sense and is more like noise if compared to **Figure 2.B.** which has more of a streaking particle image, with its spine being more well defined. Regarding their resnorm values, **Figure 2.A.** has a much smaller resnorm value than **Figure 2.B.**, and their resnorm values respectively are 2.5 and 11.4. But with manual inspection, it cannot be justified if **Figure 2.A.** is more correct than **Figure 2.B.** because it is not a streak particle.

In this research, more detailed observations have been made of the resnorm values. It is shown that streak particles which have a bigger pixel area are more prone to bigger resnorm values (which meant bigger error values). In this part, more numerical details will be shown about the resnorm values which have been grouped to separate groups based on their area range classifications.



```
+4   Local.LocalMin   allArray
Local.LocalMin
Field ▲        Value
min1           11.4110
min2           [ ]
min3           0.6043
min4           0.9795
min5           0.0515
min6           0.0107
```

```
Global
1x1 struct with 5 fields
Field ▲         Value
allArray        1x75 double
GlobalAvg       0.9737
GlobalMedian    0.1276
GlobalMax       13.6570
GlobalMin       0.0107
```

```
Local.LocalAvg
Local.LocalAvg
Field ▲        Value
avgA1          12.5340
avgA2          NaN
avgA3          5.5347
avgA4          1.4575
avgA5          0.6670
avgA6          0.0730
```

```
ResnormLists
1x1 struct with 6 fields
Field ▲        Value
x1             [11.4110,13.6570
x2             [ ]
x3             [8.1039,10.6071,.
x4             [0.9795,1.1632,2.
x5             1x28 double
x6             1x38 double
```

```
Local.LocalMax
Local.LocalMax
Field ▲        Value
max1           13.6570
max2           [ ]
max3           10.6071
max4           2.2297
max5           6.2387
max6           0.2328
```

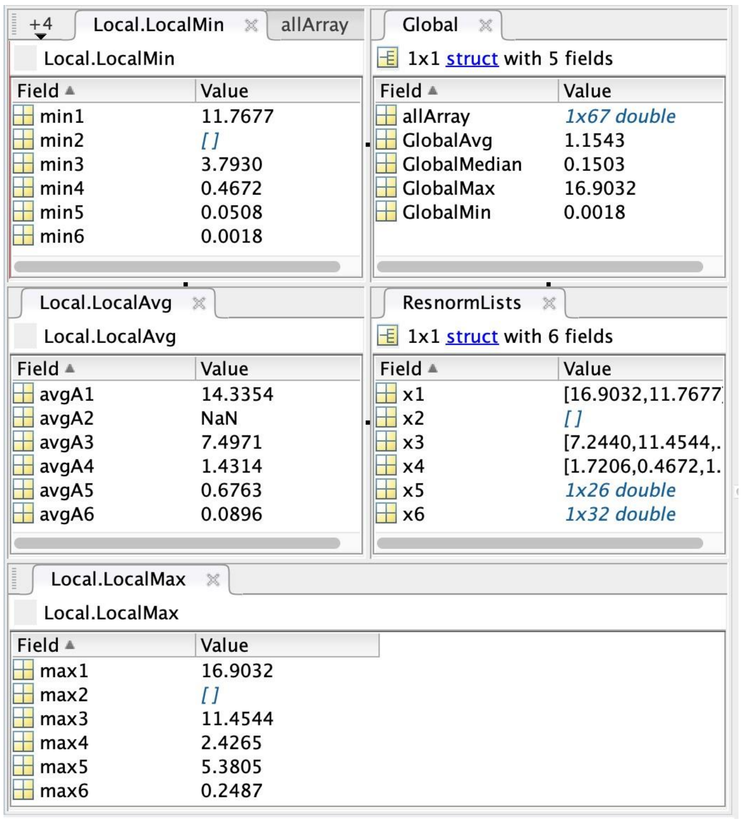
Figure 3.A. "ResnormStatistics.m" results in the MATLAB workspace tab.

Figure 3.A. shows an example of the "ResnormStatistics.m" program's results. It is shown that it holds the statistical information of average, maximum, minimum, and mean values of the streaks' resnorm values based on each group, or even all together. This specific program is used after obtaining the required data from running the "streakprocessv2r1.m" program. The data are then calculated with the formulas that are inside the "ResnormStatistics.m" program.

As explained in the paragraph above, there are 6 categorized population groups based on their pixel area. Taking the "Local.LocalAvg.avgA1" as an example, it means that the population group of Group 1 (100 pixels and above), has an **average resnorm value** of

12.5340. The second example is taking the "Local.LocalMax.max4" variable, which means that **2.2297** is the **maximum resnorm value** of Group 4 (40-59 pixels group). The Global structure has a content of the global average, median, maximum, and minimum resnorm values with all the streak's resnorm values listed on one variable ("Global.allArray").

In addition, the 4 figures below (**Figures 3.B. to 3.E.**) will show different results based on the processed images. The target for showing the other figures is to show the diversity of the processing results.



Local.LocalMin

Field	Value
min1	11.7677
min2	[]
min3	3.7930
min4	0.4672
min5	0.0508
min6	0.0018

1x1 struct with 5 fields

Field	Value
allArray	1x67 double
GlobalAvg	1.1543
GlobalMedian	0.1503
GlobalMax	16.9032
GlobalMin	0.0018

Local.LocalAvg

Field	Value
avgA1	14.3354
avgA2	NaN
avgA3	7.4971
avgA4	1.4314
avgA5	0.6763
avgA6	0.0896

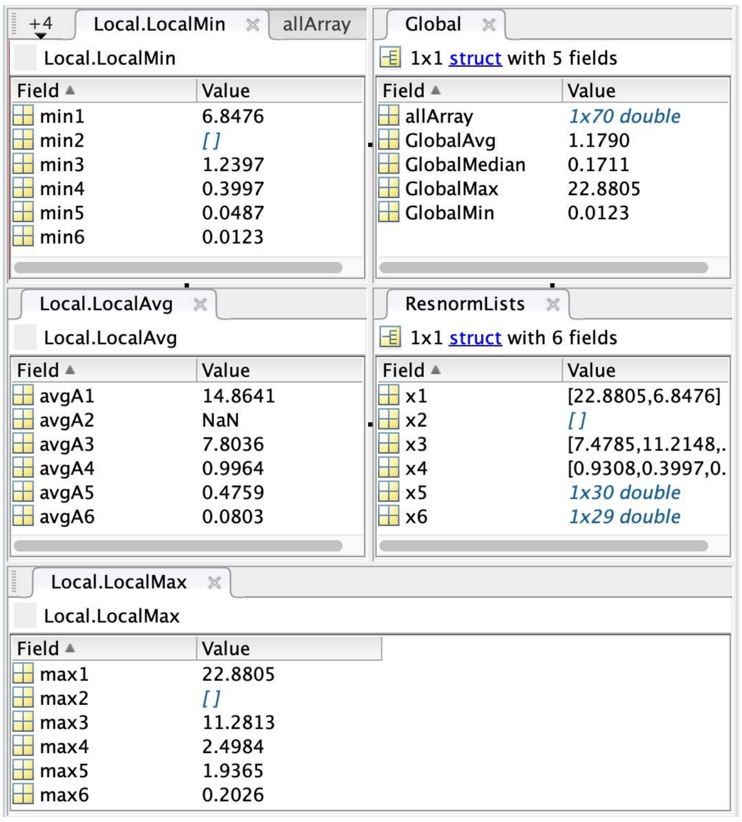
1x1 struct with 6 fields

Field	Value
x1	[16.9032,11.7677
x2	[]
x3	[7.2440,11.4544,.
x4	[1.7206,0.4672,1.
x5	1x26 double
x6	1x32 double

Local.LocalMax

Field	Value
max1	16.9032
max2	[]
max3	11.4544
max4	2.4265
max5	5.3805
max6	0.2487

Figure 3.B. "ResnormStatistics.m" second results.



Local.LocalMin

Field	Value
min1	6.8476
min2	[ ]
min3	1.2397
min4	0.3997
min5	0.0487
min6	0.0123

1x1 struct with 5 fields

Field	Value
allArray	1x70 double
GlobalAvg	1.1790
GlobalMedian	0.1711
GlobalMax	22.8805
GlobalMin	0.0123

Local.LocalAvg

Field	Value
avgA1	14.8641
avgA2	NaN
avgA3	7.8036
avgA4	0.9964
avgA5	0.4759
avgA6	0.0803

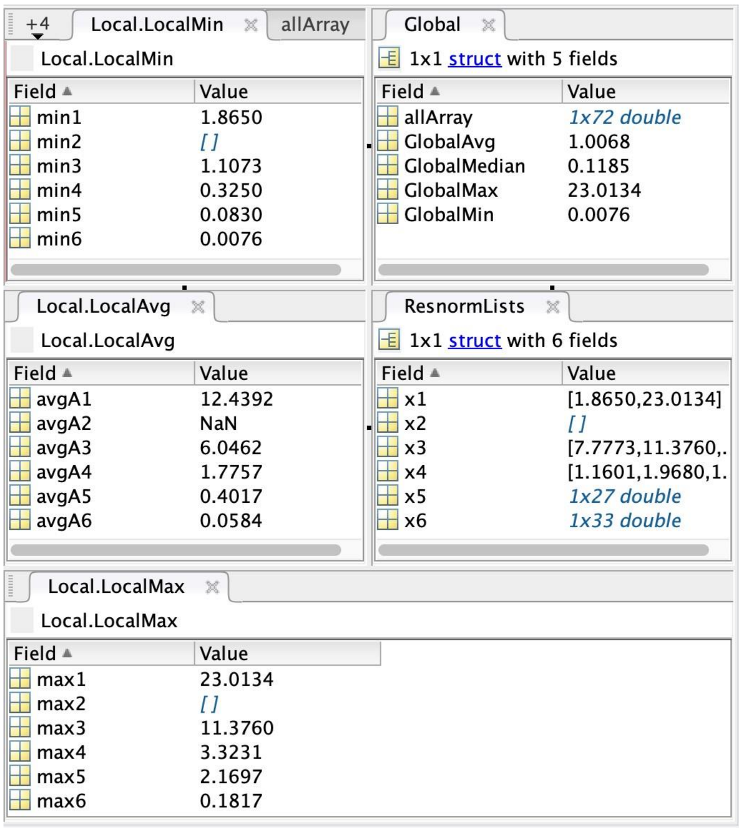
1x1 struct with 6 fields

Field	Value
x1	[22.8805,6.8476]
x2	[ ]
x3	[7.4785,11.2148,.
x4	[0.9308,0.3997,0.
x5	1x30 double
x6	1x29 double

Local.LocalMax

Field	Value
max1	22.8805
max2	[ ]
max3	11.2813
max4	2.4984
max5	1.9365
max6	0.2026

Figure 3.C. “ResnormStatistics.m” third results.



Local.LocalMin

Field	Value
min1	1.8650
min2	[ ]
min3	1.1073
min4	0.3250
min5	0.0830
min6	0.0076

1x1 struct with 5 fields

Field	Value
allArray	1x72 double
GlobalAvg	1.0068
GlobalMedian	0.1185
GlobalMax	23.0134
GlobalMin	0.0076

Local.LocalAvg

Field	Value
avgA1	12.4392
avgA2	NaN
avgA3	6.0462
avgA4	1.7757
avgA5	0.4017
avgA6	0.0584

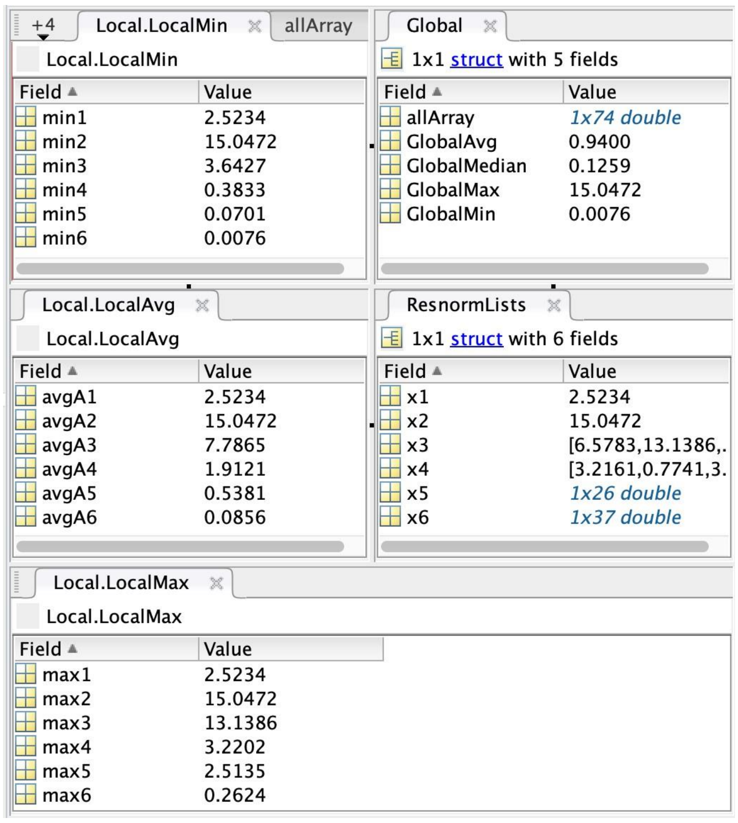
1x1 struct with 6 fields

Field	Value
x1	[1.8650,23.0134]
x2	[ ]
x3	[7.7773,11.3760,.
x4	[1.1601,1.9680,1.
x5	1x27 double
x6	1x33 double

Local.LocalMax

Field	Value
max1	23.0134
max2	[ ]
max3	11.3760
max4	3.3231
max5	2.1697
max6	0.1817

Figure 3.D. “ResnormStatistics.m” fourth results.



Local.LocalMin

Field	Value
min1	2.5234
min2	15.0472
min3	3.6427
min4	0.3833
min5	0.0701
min6	0.0076

1x1 struct with 5 fields

Field	Value
allArray	1x74 double
GlobalAvg	0.9400
GlobalMedian	0.1259
GlobalMax	15.0472
GlobalMin	0.0076

Local.LocalAvg

Field	Value
avgA1	2.5234
avgA2	15.0472
avgA3	7.7865
avgA4	1.9121
avgA5	0.5381
avgA6	0.0856

1x1 struct with 6 fields

Field	Value
x1	2.5234
x2	15.0472
x3	[6.5783,13.1386,.
x4	[3.2161,0.7741,3.
x5	1x26 double
x6	1x37 double

Local.LocalMax

Field	Value
max1	2.5234
max2	15.0472
max3	13.1386
max4	3.2202
max5	2.5135
max6	0.2624

Figure 3.E. “ResnormStatistics.m” fifth results.

Chapter II: Hardware Setups for Gradient Illumination Technique

Some additional hardware is needed to make this step achievable. The first is to add an Arduino Uno board, so that the board may be programmed to output signals to the LED, and to the camera. The problem is that the Arduino board cannot connect directly to the LED lights, so a PicoBuck LED Driver is used to become a mediator between the two parts. Then with this, the Arduino will send a signal that will trigger the camera's exposure start and the LED brightness intensity as well.

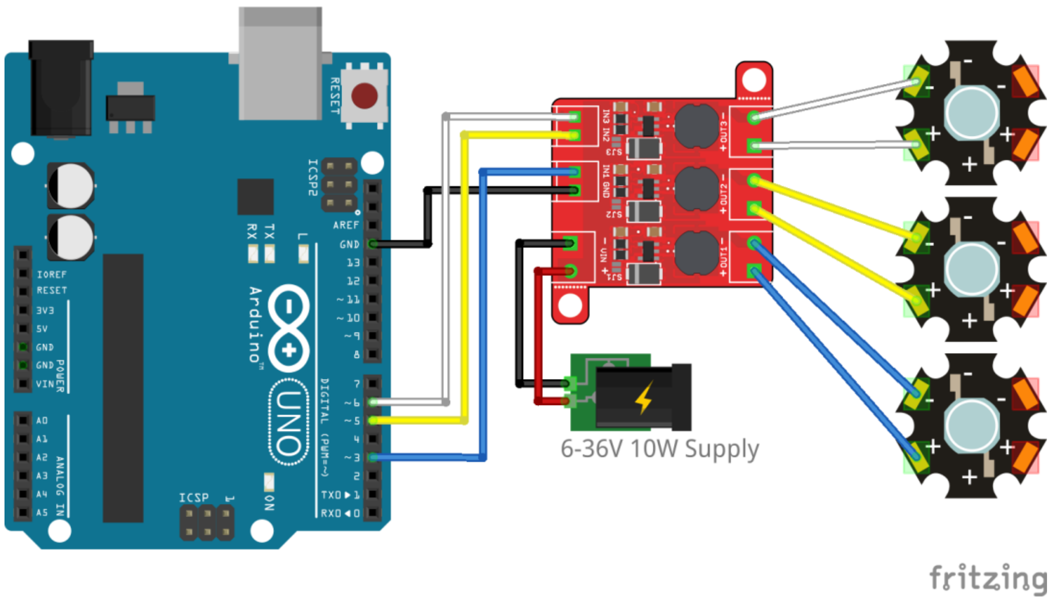


Figure 4.A. Arduino connection to PicoBuck driver and LED schematics (Source: https://learn.sparkfun.com/tutorials/picobuck-hookup-guide-v12/all).

With the current setup, the Arduino was designed to be in the master mode, and the camera in the slave mode (Arduino initiates the trigger signal, and the camera must wait until it receives the signal so that it can start its exposure time). The Arduino will have 2 cables that are connected to the camera's input and ground slot. The Arduino's slot number (**~9**) will be where the output cable from the Arduino into the input slot of the camera be inserted. Then, any ground slot of the Arduino must also be connected to the camera's ground slot. The setup also includes some delays that are caused by the internal systems and basic physics law. For example, the camera needs some delay time after receiving the trigger signal from the Arduino, to start capturing and start the exposure. The camera also has a data readout delay time and may be adjusted accordingly as well.

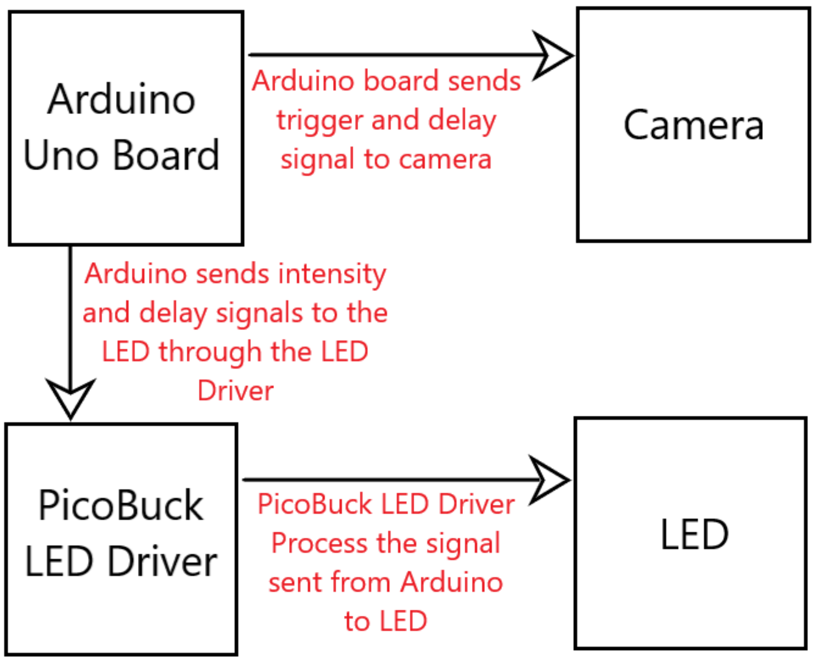

Arduino Uno Board
Arduino board sends trigger and delay signal to camera
Camera
Arduino sends intensity and delay signals to the LED through the LED Driver
PicoBuck LED Driver
PicoBuck LED Driver Process the signal sent from Arduino to LED
LED

Figure 4.B. The hardware setup schematic diagram.

The camera type used in this research was the **Ximea MQ013CG-ON.** The readout delay time of this camera, however, is still unknown because the camera that was used does not have the manual to calculate the readout time. So, this research used 7000 microseconds as the readout delay time. The results turn out to be good and are still as expected because the observable particle streaks still have the dim to the bright intensity gradient. For the image capturing method of the gradient illuminated particles, the magnification used was around 20x and the exposure time used was around 24.99 milliseconds. In these 24.99 milliseconds, the light intensity value goes up from 1 up to 255.

The Arduino code is designed to send the trigger signal with the shape of a rising edge. The rising edge high time lasts for 10 microseconds and then is continued directly by the exposure of the image for 24990 microseconds. While this exposure time is running, the LED intensity starts to grow from 1 to 255. The number of 1 to 255 is obtained from the Arduino analog values for the minimum and maximum values, where the analog value of 0 means 0 volts, the analog value of 255 means 5 volts, and the analog values between 0-255 are linearly correlated from 0 to 5 volts. The complete Arduino code will be included in the attachment section.

A specific LED angle needs to be determined by the researcher, such that the light will shine on the particles but leaves the background dark enough to leave a good amount of contrast between the particle and the background. The focus length of the LED needs to be determined as well so that the focus lands right on the particle's layer. With this, it is recommended that the light is not directly above the particle and shines directly to the camera underneath. The schematic of this problem will be shown in **Figure 4.C.** below.

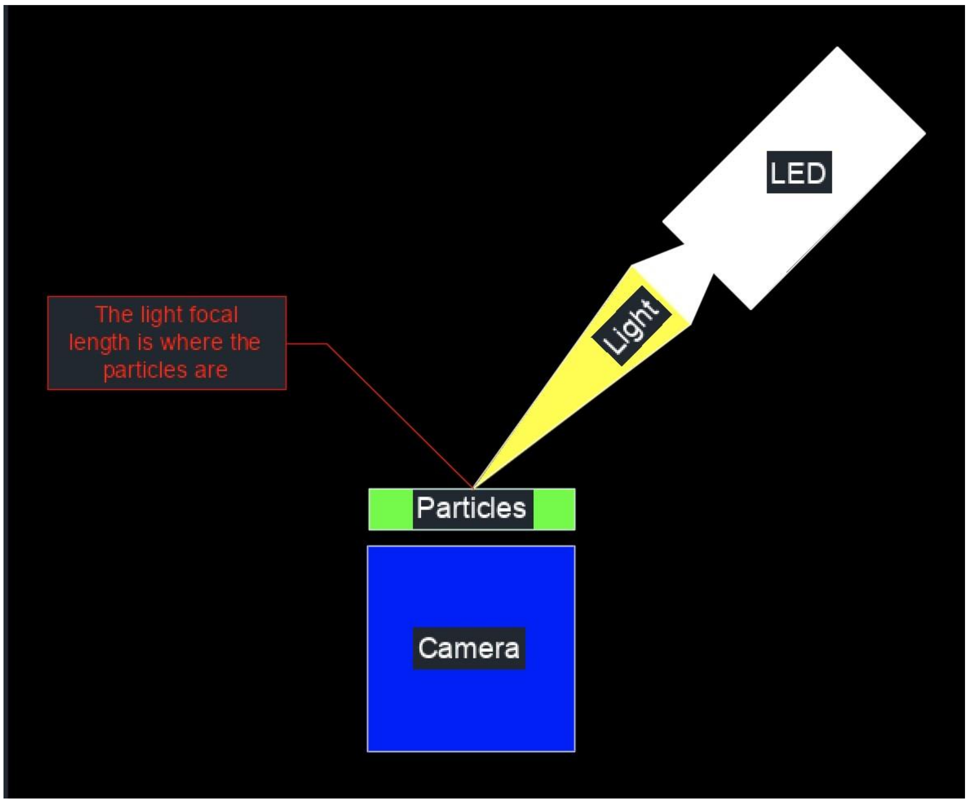


Figure 4.C. LED, camera, and particle setup diagram.

The particles that are used as experimental objects are placed inside a microchannel mold. Before injecting the particles into the mold, the mold must be cleaned with alcohol and vibrated in the vibrator box so that it may be clean for use. After the channel inside the microchannel is cleaned, the surface glass of the microchannel must also be cleaned with alcohol and dried by pressured air sprayer. After all of these steps are done, the microchannel can be inserted by the particles.

The method that is used to move the particles was to put an injection in one end of the microchannel, then inject the microchannel with more particles so that the camera will capture the moving particles due to injection pressure from the upcoming particles. However, the downside of the injection method is that the researcher must press the injector in a very gentle way because even a very small push force will accelerate the particles fast enough such that the camera could not catch the particles' streak image.

Besides using the injection to move the particles, the use of piezoelectric vibrations to move the particles is a better way to obtain clearer experimental images. The piezoelectric method is a better way to deliver an adequate amount of vibration forces, such that the particles can move around the microchannel at a good pace. Firstly, the piezoelectric is connected to the signal generator. The frequency and the voltage of the signal generator can be adjusted accordingly depending on the researcher's goal (12 kHz and 20 Volts in this experiment). Then, the piezoelectric can be placed on top of the microchannel and stick to the top of the microchannel surface with the help of lubricant gel, nearby its small triangular mold shapes so that the vibrations will be near enough to the particles inside the microchannel.

One of the problems encountered during the setting up of the hardware is that the camera manual doesn't have the data readout delay formula for this particular camera. So, the researchers used a guessed data readout delay number of 7000 microseconds, which had been explained in the paragraph above. With this problem in hand, the readout time may interrupt the perfect synchronization of the camera and the blinking of the LED light, so that the initial light value during the start of the exposure time is not the value set for the start of the exposure time (i.e., At the start of the exposure time, the intensity must be "1" but for some reason of non-perfect synchronization, the intensity value during the exposure start is not 1).

Chapter III: Gradient-Illuminated Experimental Images

With the new hardware setups, some newer experimental images with the gradient illumination technique had been captured. Some of the captured experimental images will be shown in **Figure 5.A.** until **5.C.** below.

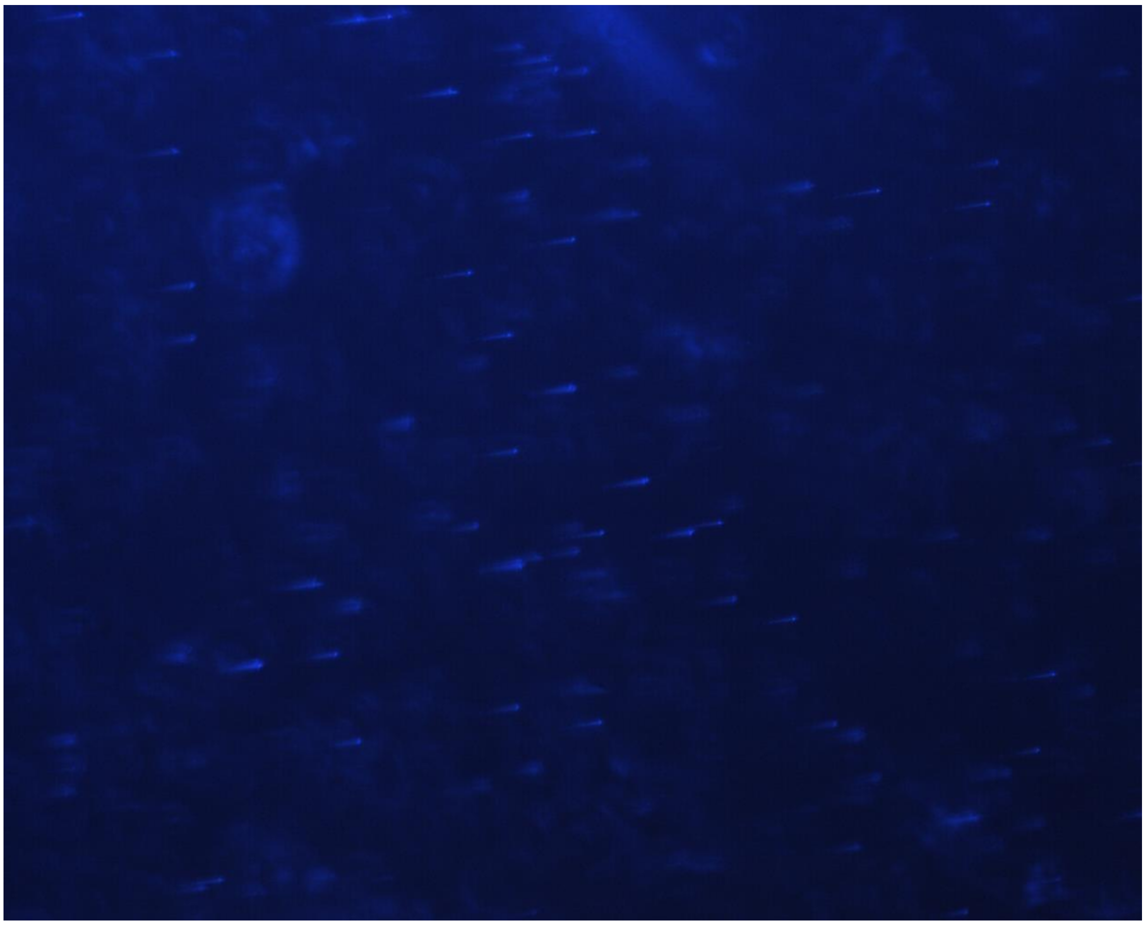


Figure 5.A. First gradient illumination experimental image.

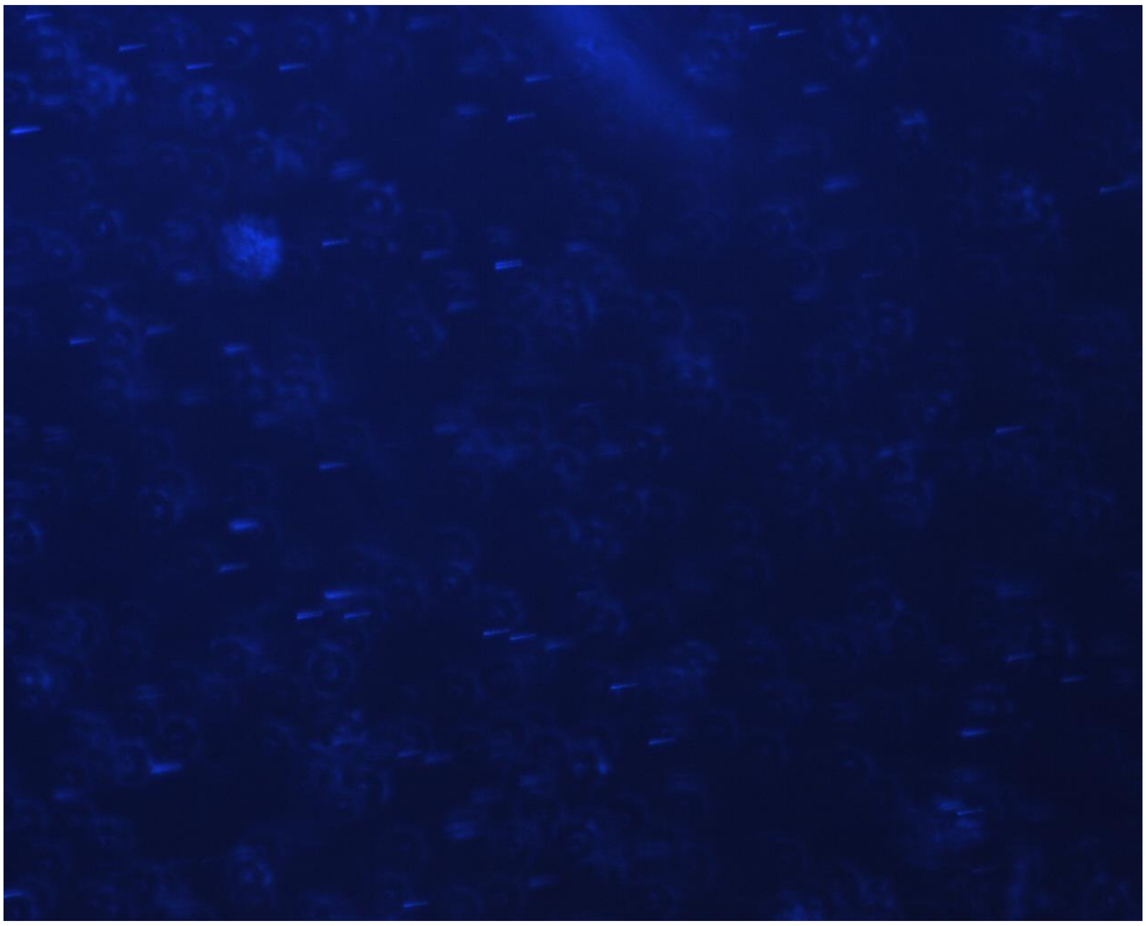


Figure 5.B. Second gradient illumination experimental image.

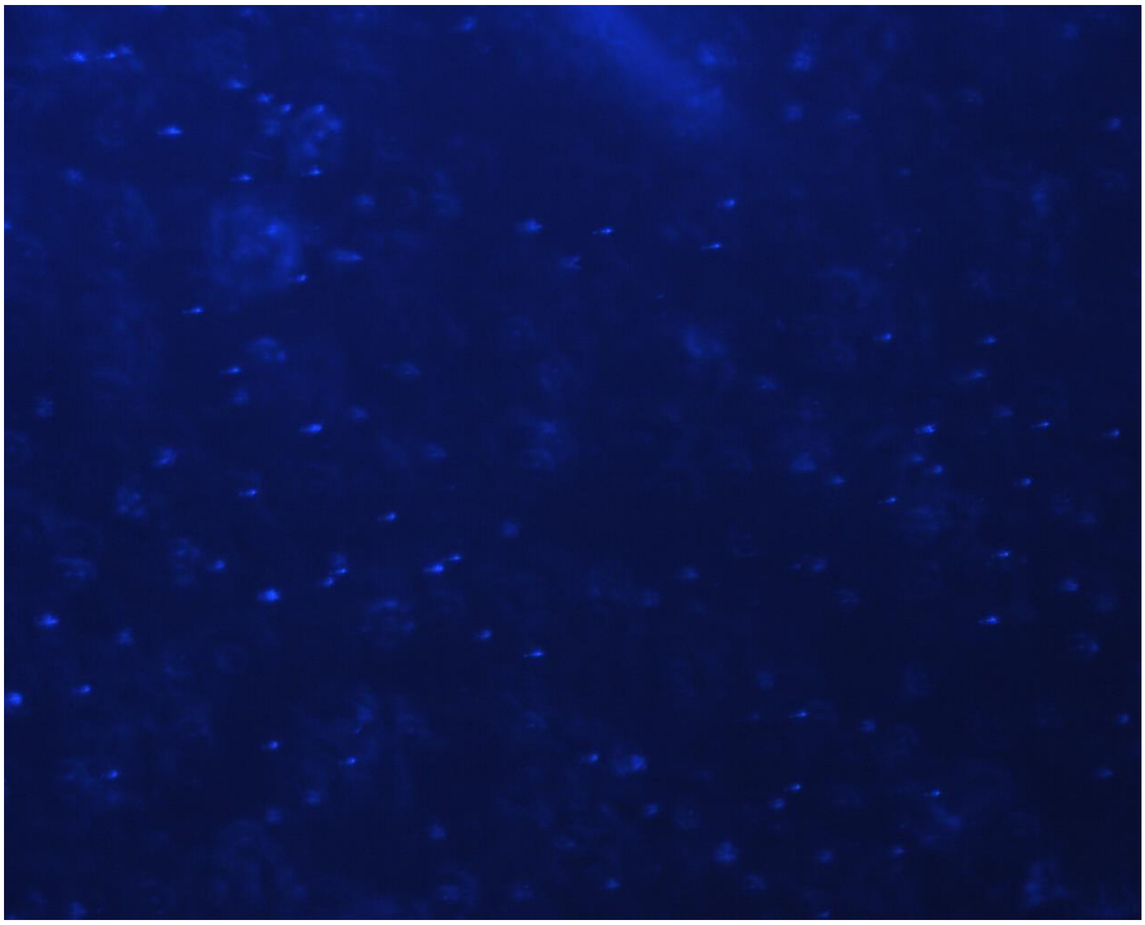


Figure 5.C. Third gradient illumination experimental image.

From the three experimental images shown above, we can see that the background has no direct light source that will outshine the streak particles. The light angle shines just an adequate amount of lighting, and the streak particles have a bright view because they are reflecting the light. However, we can see that there are a lot of noise particles. This is due to the streak particles that are in the micro-channel, which are also in different layer depths. The lenses can only focus on one layer at a time so that the particles in different layer depths are unfocussed, causing the circular noise to blur across the image.

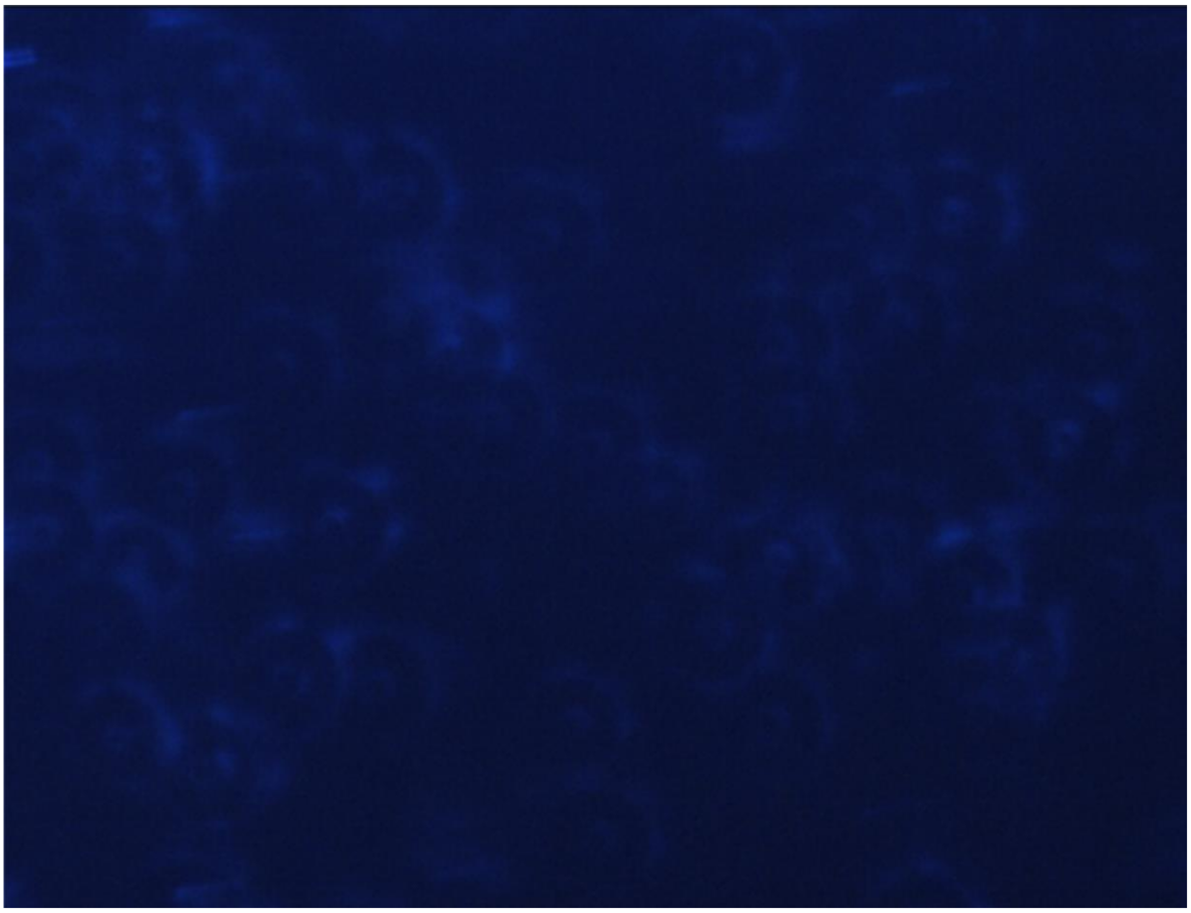
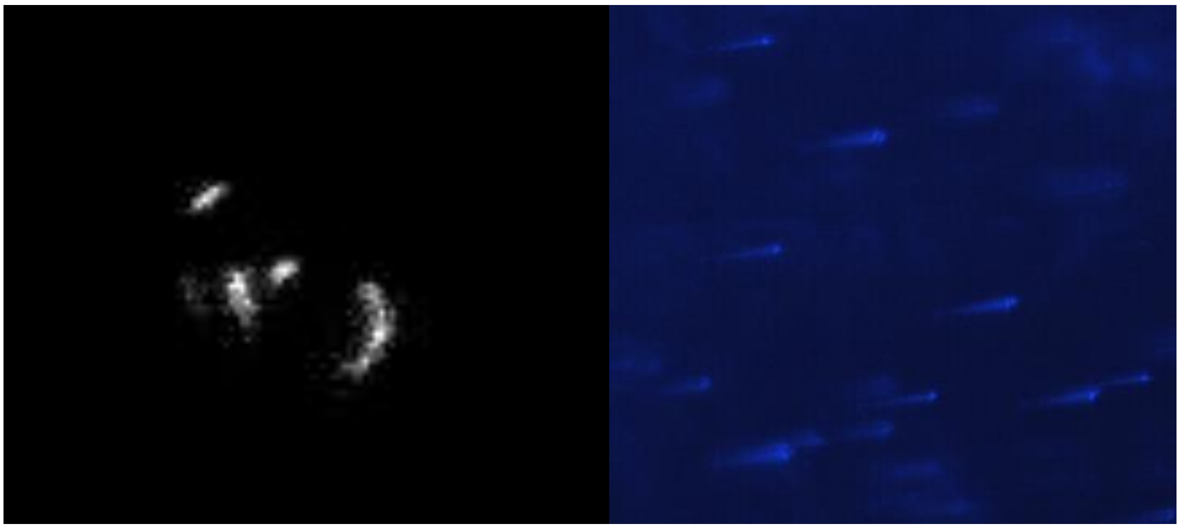


Figure 5.D. Zoomed-in picture of figure 2.B.'s circular noises.

The newest results that were obtained from the new setup seem to be working as planned. The sample images show that there are light intensity gradients along with the streaking particles. The technique that was used to do this intensity variation experiment is that the light intensity will keep on increasing until its maximum intensity for the exposure's time duration. At the start of the exposure, its intensity is at its minimum, and it gradually increases and reaches the peak intensity at the end of the exposure time.

We can see from above that the trailing streak leaves a darker trail than the one ahead of it, which has brighter intensity. This method of experimental image acquisition cannot be achieved by the old image capturing method, and this method is available because the LED light's intensity can be scaled and adjusted accordingly with the help of the PicoBuck LED Driver and Arduino Uno board. **Figure 5.E.** shown below will show the difference between the old captured experimental streak images and the new ones.

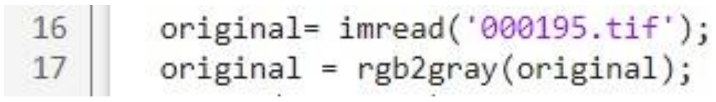


(a) **(b)**

Figure 5.E. (a) & (b) Comparison of the (a)old and (b)new experimental images.

The older particle streak image shows that it doesn't present a good gradient of intensity, so it is definitely hard to see which one is the starting, middle, and end points of those streaks. Unlike it, the newly captured experimental streak images are showing us with bare eyes the starting and ending points of those particle streaks. The darker end of the streak indicates the starting point, and the bright peak on the opposite end means the ending/final point of the streak.

The image processing part with the MATLAB algorithm, this part has not been achieved yet due to time limitations. The program has been pre-made, and the program was not runnable because it still needs some editing and adjustments. The program can only run a grayscale image, so **Figure 5.E.(b)** image cannot be processed by the program yet. With the help of MATLAB's command of "i.e., image = rgb2gray("image_file");", the blue-colored image will be converted into a gray-scaled image.



```
original= imread('000195.tif');
original = rgb2gray(original);
```

Figure 5.F. Conversion of RGB image into grey-scaled image code.

Chapter IV: Future Works

The current research has almost achieved all of the hardware setup requirements, except for the data readout time delay value. In the future, the readout delay time must be defined correctly such that the camera and LED’s blink sync well. Also, it is suggested that before picking the camera for the experiment, researchers must research the camera’s specifications completely so that the research may be done perfectly. The perfect synchronization between the camera and the LED light blinking time is crucial for the image-taking process. The accuracy of the delay time is up to the microseconds level so a slight error can cause huge and incrementing error values.

In the future, the program that is developed for this newly developed gradient illuminated experimental image should be edited accordingly, optimized to have a shorter runtime, and be tested to know whether the program works as expected, or not. The program has not been edited nor tested due to time limitation reasons, so the results of running the program have not been studied yet.

Conclusion

With the newly developed image capturing method that implements the use of light intensity variation, researchers now may correctly define the starting, middle, and end-points of the streaking particles; unlike the previous experimental image capturing method. Particle Streak Velocimetry may be an opportunity door for those who do not have any good hardware equipment but needs to do the particle velocimetry experiments. The captured image has shown us some great prospects that it may contribute to future research, and with the right algorithm development to resolve the image, it can be a reliable source and method of velocimetry technique as well.

References

I) Mumtaz Hussain QURESHI and Wei-Hsin TIEN - *A Novel Streak -Resolving Algorithm for Particle Streak Velocimetry*

II) Mumtaz Hussain QURESHI , Wei-Hsin TIEN and Yi-Jiun Peter LIN - *Performances Comparison of Particle Tracking Velocimetry (PTV) and Particle Image Velocimetry (PIV) with Long Exposure Particle Streaks*

Attachments

1. streakprocessV2r1.m code

```
function Results=streakprocessV2r1(filename)
close all;
warning ('off');
original = imread(filename);
image_type = class(original);

se = strel('disk',5);
tophatFiltered = imtophat(original,se);

I2=tophatFiltered;
% Set threshold to further ignore any low intensity pixel
thr=50;

% Apply the thresholding
I3=I2.*I2>thr;

% Define blobs by checking the connection of pixels
I3_cc=bwconncomp(I3);

% Original image masked by the tophatfilter + thresholding
if image_type == "uint8"
    I3_gray=original.*uint8(I3);
elseif image_type == "uint16"
    I3_gray=original.*uint16(I3);
elseif image_type == "uint32"
    I3_gray=original.*uint32(I3);
else
    return
end
% Use "regionprops" to collect more blob properties
I3_stats1 = regionprops(I3_cc,'Centroid','PixelList',...
    'BoundingBox','Area','Image');
%%
for i=1:size(I3_stats1,1)
    BBox=I3_stats1(i).BoundingBox;
    grayI=I3_gray(ceil(BBox(2)):ceil(BBox(2))+BBox(4)-
1,ceil(BBox(1)):ceil(BBox(1))+BBox(3)-1);
    I3_stats1(i).Blob=grayI;
end
% Filter out particle image that has smaller area than thr_area
thr_area=12;


%%%%%%%%%%%%%%%%%%%  CURVEFITTING PROCESS %%%%%%%%%%%%%%%%%%%%%%%

idx = [I3_stats1.Area]>=thr_area; %seeking logical values of 0 and 1
idx=reshape(idx,[],1);
a=(1:length(idx))';
b=a.*idx;
Blob_filtered_idx=b(all( b,2),:);

counter=1;
for n=1:length(Blob_filtered_idx)
 Blob_filtered_idxx = Blob_filtered_idx(n);
 Blob_filtered = I3_stats1(Blob_filtered_idxx).Blob;
 PixelListt = I3_stats1(Blob_filtered_idxx).BoundingBox;
 PixelListt = [PixelListt(1) PixelListt(2)];
Blob = Blob_filtered;
 [rr, cc] = size(Blob);
r=round(rr/2);
c=round(cc/2);
xdata_idx=find(I3_stats1(Blob_filtered_idxx).Image==1);
[xdata(:,1),xdata(:,2)]=find(I3_stats1(Blob_filtered_idxx).Image==1);
A=Blob(xdata_idx);
I3_stats1(Blob_filtered_idxx).I3_Region=[xdata,A];
A=double(A);
if image_type == "uint8"
    A=A./255;
elseif image_type == "uint16"
    A=A./65535;
elseif image_type == "uint32"
    A=A./4294967295;
else
    return
end

ydata=A; % BLOB consisted of 0 matrixes with size of 193 x 193 double.

x001=[0.5, r/2,   0.9,      c/2,     1,     8e-10 ,    2.7e-09,        3.4e-09,
9.5e-07,    1.5e-05,       0.007,         0.01,      0,   0.00001 ,     0.00001,
0.00001,      0.00003,            0.00007,       0.07,       0.01,     0];

lb= [0.1,    0,   0.8,         0,     1,   -7.2e-10,     -2.6e-09,       -3.3e-09,
-9.4e-07,   -1.4e-05,      -0.06,         -0.09,        0 ,  0.00001,    0.00001,
0.00001,      -0.00002,           -0.00006 ,     -0.06,     -0.09,     0];
ub=[1.1,     r,   1.2,     c,     1,     7.2e-10,    2.6e-09,        3.3e-09,
9.4e-07,    1.4e-05,        0.06,          0.09,        193 ,  0.00001,    0.00001,
0.00001,       0.00002,            0.00006,       0.06,       0.09,     193];


[x1,resnorm1]= lsqcurvefit(@D2GaussFunction2,x001,xdata,ydata,lb,ub);

Fittedanswers1 = x1;
Error1 = resnorm1;

x002=[0.5, r/2-2,   0.9,    c/2-2,   1,   4e-09 ,   1.2e-08,   3.3e-09,    2.1e-
06,        6e-05 ,  0.0095,    5 ,   0,    3e-10 ,   8e-10,   5e-08,       10e-07,
0.0004,   0.0095,     5,     0];

lb= [0.1,     0,   0.8,    0,    1 , -3e-09,    -1.1e-08,   -3.2e-09,  -2.05e-06,
-5.5e-05, -0.0095,   -4,    0 ,  -2e-10,  -7e-10,  -4e-08,    -9.99e-07,    -
0.0003,   -0.0095,    -4,    0];
ub=[1.1,    r,  1.2,  c,    1,    3e-09,    1.1e-08,    3.2e-09,   2.05e-06,
5.5e-05,  0.0095,    4,     193 ,  2e-10,    7e-10,  -4e-08,    -9.99e-07,
0.0003,    0.0095,     4,    193];

[x2,resnorm2]= lsqcurvefit(@D2GaussFunction2,x002,xdata,ydata,lb,ub);

Fittedanswers2 = x2;
Error2 = resnorm2;
```

```
x003=[0.5,  r/2-4,   0.9,   c/2-4  , 1,   10e-08,  6e-07,    6e-05,   0.006,
0.09,     0.7,     37,   0,   4e-08,     3e-07,      4e-06,   3e-04,     0.07,
0.7 ,    21,   0];

lb= [0.1,     0,    0.8,     0  ,  1 , -9e-08,  -5e-07,   -5e-05,   -0.005,    -
0.09,   -0.6,   -36,  0, -3e-08,    -2e-07,    -3e-05,   -2e-04,    -0.06,   -
0.6  ,   -20,  0];
ub=[1.1,    r,   1.2,   c , 1,   9e-08,   5e-07,     5e-05,    0.005,     0.09,
0.6,    36,  193,   3e-08,     2e-07,     3e-05,     2e-04,     0.06,     0.6,
20,  193];

[x3,resnorm3]= lsqcurvefit(@D2GaussFunction2,x003,xdata,ydata,lb,ub);

Fittedanswers3 = x3;
Error3 = resnorm3;


x004=[0.5,  r/2-6,   0.9,  c/2-6,    1,       4e-08,     3e-07,      4e-06,   3e-
04,     0.07,       0.7 ,    21,   0,  10e-08,  6e-07,    6e-05,   0.006,
0.09,     0.07,   37,   0];

lb= [0.1,      0,   0.8,   0,     1,    -3e-08,    -2e-07,    -3e-05,   -2e-04,
-0.06,   -0.6  ,   -20,  0,   -9e-08,  -5e-07,   -5e-05,   -0.005,    -0.09,   -
0.06,    -36,  0];
ub=[1.1,     r,   1.2,  c  , 1,     3e-08,     2e-07,     3e-05,     2e-04,
0.06,     0.6,     20,  193,     9e-08,   5e-07,    5e-05,    0.005,     0.09,
0.06,    36,  193];

[x4,resnorm4]= lsqcurvefit(@D2GaussFunction2,x004,xdata,ydata,lb,ub);

Fittedanswers4 = x4;
Error4 = resnorm4;


x005=[0.5, r/2-6, 0.9,  c/2-6  , 1,      10e-08,   6e-07,   6e-05,   0.006,
0.009 ,   0.7 ,  13,   0,   4e-08,    3e-07,   4e-05,   3e-04,  0.07,  0.7 ,
13,   0];

lb= [0.1,   0,   0.8,    0  ,  1 ,  -9e-08,  -5e-07,   -5e-05, -0.005,   -0.0099,
-0.6,  -12,   0,   -3e-08,  -2e-07,  -3e-05,  -2e-04, -0.06,  -0.6,  -12,  0];
ub=[1.1,  r ,   1.2,  c  , 1,   9e-08,   5e-07,    5e-05,   0.005,    0.0099,
0.6,   12,   193,    3e-08,   2e-07,    3e-05,   2e-04,  0.06,   0.6,    12,
193];

[x5,resnorm5]= lsqcurvefit(@D2GaussFunction2,x005,xdata,ydata,lb,ub);

Fittedanswers5 = x5;
Error5= resnorm5;

x006=[0.5,  r/2-6,   0.9,   c/2-6  , 1,   10e-08,  6e-07,     6e-05,     7e-04,
0.05,       0.7,    37,   0,   4e-08,     3e-07,      4e-06,        3e-04,
0.08,     0.08 ,    37,   0];

lb= [0.1,     0,    0.8,     0  ,  1 , -9e-08,  -5e-07,   -5e-05,   -6e-04,    -
0.04,   -0.6,    -36,     0,   -3e-08,    -2e-07,    -3e-05,     -2e-04,     -
0.07,    -0.07  ,   -36,  0];
ub=[1.1,     r,   1.2,    c  , 1,   9e-08,   5e-07,    5e-05,    6e-04,     0.04,
0.6,    36,     193,     3e-08,     2e-07,     3e-05,     2e-04,      0.07,
0.07,      36,  193];
```

```
[x6,resnorm6]= lsqcurvefit(@D2GaussFunction2,x006,xdata,ydata,lb,ub);

Fittedanswers6 = x6;
Error6 = resnorm6;


x007=[0.5,  r/2-6,   0.9,     c/2-6,    1,   4e-08,     3e-07,      4e-06,     3e-
04,     0.08,     0.08 ,    37,     0,   10e-08,  6e-07,     6e-05,     7e-04,
0.05,   0.7,    37,   0];

lb= [0.1,     0,    0.8,     0,   1,  -3e-08,    -2e-07,    -3e-05,     -2e-04,
-0.07,    -0.07,    -36,    0,   -9e-08,  -5e-07,   -5e-05,   -6e-04,    -0.04,
-0.6,   -36,  0];
ub=[1.1,     r,   1.2,   c  , 1,   3e-08,     2e-07,     3e-05,      2e-04,
0.07,     0.07,     36,    193,     9e-08,   5e-07,    5e-05,     6e-04,     0.04,
0.6,    36,  193];

[x7,resnorm7]= lsqcurvefit(@D2GaussFunction2,x007,xdata,ydata,lb,ub);

Fittedanswers7 = x7;
Error7 = resnorm7;


x008=[0.5, r/2-2,   0.9,    c/2-2,    1,   4e-09 ,   1.2e-08,   3.3e-09,    2.1e-
06,        6e-05 ,      0.0095,    2 ,   0,    3e-10 ,   8e-10,   5e-08,
10e-07,    0.0004 ,   0.0095 ,        2 ,   0]

lb= [0.1,     0,   0.8,    0,   1, -3e-09,    -1.1e-08,   -3.2e-09,  -2.05e-06,
-5.5e-05,   -0.0095,   -1,    0 ,  -2e-10,  -7e-10,  -4e-08,    -9.99e-07,  -
0.0003 ,  -0.0095,      -1,    0];
ub=[1.1,    r,  1.2,  c,   1,    3e-09,    1.1e-08,    3.2e-09,   2.05e-06,
5.5e-05,    0.0095,    1,     193 ,  2e-10,    7e-10,  -4e-08,    -9.99e-07,
0.0003,    0.0095,      1,    193];

[x8,resnorm8]= lsqcurvefit(@D2GaussFunction2,x008,xdata,ydata,lb,ub);

Fittedanswers8 = x8;
Error8 = resnorm8;

resnorm = [resnorm1, resnorm2, resnorm3, resnorm4 ,resnorm5,resnorm6,
resnorm7 ,resnorm8]; x = [x1; x2; x3; x4; x5;x6; x7; x8 ]; [~, idx] =
min(resnorm);
Fittedanswers = x(idx,:);
Error = resnorm(idx);
Fitteddata(n,1:21) = Fittedanswers;
Fitteddata(n, 22) = Error;

px_fit(n,1:8)=Fittedanswers(:,14:21);
x_coef=px_fit(n,1:8);
x_initial_blob=Fittedanswers(:,4);
px_der_fit(n,:)=polyder(px_fit(n,1:8));

py_fit(n,1:8)=Fittedanswers(:,6:13);
y_coef=py_fit(n,1:8);
y_initial_blob=Fittedanswers(:,2);
py_der_fit(n,:)=polyder(py_fit(n,1:8));

% additional parts for streak locations
```

```
x_initial_shift=x_initial_blob+PixelListt(1);
y_initial_shift=y_initial_blob+PixelListt(2);
x_initial1(n) = polyval(x_coef,0)+x_initial_shift;
y_initial1(n) = polyval(y_coef,0)+y_initial_shift;

x_middle1(n) = polyval(x_coef,0.5)+x_initial_shift;
y_middle1(n) = polyval(y_coef,0.5)+y_initial_shift;

x_final1(n) = polyval(x_coef,1)+x_initial_shift;
y_final1(n) = polyval(y_coef,1)+y_initial_shift;


u_initial_pt_fit(n)=polyval(px_der_fit(n,:),0);
u_middle_pt_fit(n)=polyval(px_der_fit(n,:),0.5);
u_final_pt_fit(n)=polyval(px_der_fit(n,:),1);

v_initial_pt_fit(n)=polyval(py_der_fit(n,:),0);
v_middle_pt_fit(n)=polyval(py_der_fit(n,:),0.5);
v_final_pt_fit(n)=polyval(py_der_fit(n,:),1);

Dia_fit(n)=Fittedanswers(:,3)*2;
Amp_fit(n)=Fittedanswers(:,1);
Resnorm_List(n)=Fitteddata(n, 22);

%% Additional code for resnorm lists on the I3_stats1 table
I3_stats1(Blob_filtered_idxx).Resnorm=Resnorm_List(n);

% Data save to a structure
fprintf('Just finished iteration #%d\n', counter);
counter=counter+1;
xdata=[];
end

Results.I3_stats1=I3_stats1;
Results.Fitteddata=Fitteddata;
Results.px_fit=px_fit;
Results.px_der_fit=px_der_fit;
Results.py_fit=py_fit;
Results.px_der_fit=px_der_fit;
Results.py_der_fit=py_der_fit;
Results.x_initial1=x_initial1;
Results.y_initial1=y_initial1;
Results.x_middle1=x_middle1;
Results.y_middle1=y_middle1;
Results.x_final1=x_final1;
Results.y_final1=y_final1;
Results.u_initial_pt_fit=u_initial_pt_fit;
Results.v_initial_pt_fit=v_initial_pt_fit;
Results.u_middle_pt_fit=u_middle_pt_fit;
Results.v_middle_pt_fit=v_middle_pt_fit;
Results.u_final_pt_fit=u_final_pt_fit;
Results.v_final_pt_fit=v_final_pt_fit;
Results.Dia_fit=Dia_fit;
Results.Amp_fit=Amp_fit;
Results.Resnorm_List=Resnorm_List;


function F = D2GaussFunction2(x,xdata)
```

```
 F =integral(@(t) x(1)*exp(   -(((xdata(:,1)-x(2)-
(x(6)*t^7+x(7)*t^6+x(8)*t^5+x(9)*t^4+x(10)*t^3+x(11)*t^2+x(12)*t^1+x(13))).^2/(2*x
(3)^2) + ...
     ((xdata(:,2)-x(4)-
(x(14)*t^7+x(15)*t^6+x(16)*t^5+x(17)*t^4+x(18)*t^3+x(19)*t^2+x(20)*t^1+x(21))).^2/
(2*x(3)^2) )    ))),0,x(5),'ArrayValued',true);
```

2. BatchRun_streakprocessV2r1.m code

```
[filenameset, pathname]= uigetfile('*.tif','MultiSelect','on');
for i=1:size(filenameset,2)
    filename=[pathname filenameset{i}];
    Results=streakprocessV2r1(filename);
    save([filename '_data.mat'],'Results');
end
```

3. PlotResults.m code

```
%% RESULTS
[filename, pathname]= uigetfile('*.tif');
filename=[pathname filename];
load([filename '_data.mat']);
original=imread(filename,'tif');
u_initial_pt_fit_list=reshape(Results.u_initial_pt_fit,[],1);
u_middle_pt_fit_list=reshape(Results.u_middle_pt_fit,[],1);
u_final_pt_fit_list=reshape(Results.u_final_pt_fit,[],1);

v_initial_pt_fit_list=reshape(Results.v_initial_pt_fit,[],1);
v_middle_pt_fit_list=reshape(Results.v_middle_pt_fit,[],1);
v_final_pt_fit_list=reshape(Results.v_final_pt_fit,[],1);

x_initial=reshape(Results.x_initial1,[],1)-0.5;
y_initial=reshape(Results.y_initial1,[],1)-0.5;

x_middle=reshape(Results.x_middle1,[],1)-0.5;
y_middle=reshape(Results.y_middle1,[],1)-0.5;

x_final=reshape(Results.x_final1,[],1)-0.5;
y_final=reshape(Results.y_final1,[],1)-0.5;



Dia_fit=reshape(Results.Dia_fit,[],1);
Amp_fit=reshape(Results.Amp_fit,[],1);
Resnorm_List=reshape(Results.Resnorm_List,[],1);
plot_resnorm_idx=find(Resnorm_List>10);
scaler = double(0.5);

figure(1);
imshow(original)
hold on;

qstart=quiver(x_initial, y_initial,
u_initial_pt_fit_list*scaler,v_initial_pt_fit_list*scaler,0,'r');
xlim([0 1392]);
ylim([0 1040]);

qmid=quiver(x_middle,y_middle,u_middle_pt_fit_list*scaler,v_middle_pt_fit_list*sca
ler,0,'g');
xlim([0 1392]);
ylim([0 1040]);

qend=quiver(x_final,y_final,u_final_pt_fit_list*scaler,v_final_pt_fit_list*scaler,
0,'y');
xlim([0 1392]);
ylim([0 1040]);


xlabel('X(pixels)');
ylabel('Y(pixels)');
set ( gca, 'ydir', 'reverse' )
```

```
title('Experimental Curve-Fitted Velocity Field');
set(gca, 'FontName', 'Times')
ax = gca;
ax.FontSize = 12;
```

4. Arduino Code

```
const int CHL_1 = 3; //defining the pin numbers
const int CHL_2 = 5;
const int CHL_3 = 6;
int trigger_1 = 9;
int trigger_2 = 10;
int value = 0;
double a = 0.0960745098039216; //exp time of 24.99 ms

void setup(){
        pinMode(CHL_1, OUTPUT);
        pinMode(CHL_2, OUTPUT);
        pinMode(CHL_3, OUTPUT);
        pinMode(trigger_1, OUTPUT); //Send trigger signal from Arduino to others
        pinMode(trigger_2, INPUT);
}


void loop(){
        analogWrite(CHL_1,0);
        analogWrite(CHL_2,0);
        analogWrite(CHL_3,0);
        analogWrite(trigger_1,255);
        delayMicroseconds(10);
        analogWrite(trigger_1,0);
        for(int i=1, i<=255;i++){
                analogWrite(CHL_1,i);
                analogWrite(CHL_2,i);
                analogWrite(CHL_3,i);
                delay(a);
                }
        delayMicroseconds(7000); //the assumed readout delay time
        analogWrite(CHL_1,0);
analogWrite(CHL_2,0);
analogWrite(CHL_3,0);
analogWrite(trigger_1,255);
delayMicroseconds(10);
analogWrite(trigger_1,0);
for(int i=1, i<=255;i++){
        analogWrite(CHL_1,i);
        analogWrite(CHL_2,i);
        analogWrite(CHL_3,i);
        delay(a);
        }
delayMicroseconds(7000);
}
```