Special Topics (I) Final Report :

Post Processing of Experimental Images for Particle Streak Velocimetry

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	**:** B10735020
Department	**:** IATP - Mechanical Engineering
Supervising Professor	**:** Wei-Hsin Tien (田維欣)

Abstract

The Particle Streak Velocimetry (PSV) method is one of the methods inside the fluid mechanics field that is being used to resolve particle velocity properties during motion. The PSV method tracks every single particle; which is considered as the Lagrangian point of view. This experiment uses the PSV method to resolve the streak particle images by using the MATLAB program. The experimental image capturing is set to have a long exposure so that the captured particle image may appear to have streaks in its movement path.

The algorithm is built with MATLAB language, the main streak solving algorithm, and the streak velocity vector plotter code is designed and coded by Mumtaz Hussain Qureshi. The goal of this experiment is to run and test if the algorithm runs well based on the given experimental images. The streak-solving algorithm will process the streak images and return the U and V velocity vectors, and also other data such as the squared 2-norm of the residual list (resnorm), the coordinates/location of the streaks, diameters, also its amplitudes.

The U and V velocity vectors are separated into 3 parts, which are the initial, middle, and final velocity vectors. The elongation of the particle image due to streaking informs us that the streak was captured in a frame of exposure time. With this, the streak is divided into 3 parts so that the instantaneous velocities are shown in each of these points. The resnorm value should also be considered well because it indicates how accurate the algorithm resolves the velocity properties of the streaking particle image. The lower the resnorm value, the more accurate the resolved velocity properties are.

By utilizing the data that were achieved in the streak-resolving algorithm, we may then plot the velocity vectors so that we can map the vectors directly in the diagram, with the streak’s original locations as well. The final vector plotting results will determine whether the streak resolving algorithm does work well, or if it still needs improvement.

I. Introduction

Particle Image Velocimetry (PIV) and Particle Tracking Velocimetry (PTV) have been the two popular methods that are being used in the fluid mechanics experimental field. The main difference between these two techniques is that PIV measures a velocity field (analyzing multiple particles within the field at once), and PTV measures and tracks a single particle. These two techniques require a good high-performance camera that is able to capture a high-velocity moving object, such that the object remains still and the resulting image doesn’t contain a streaking particle image.

The PIV and PTV method requires a pair of images, which is when the particle is in the starting position ($\mathbf{t_0}$) and the particle’s final position ($\mathbf{t_1}$). The displacement is then divided by the time interval between the two image frames, and it results as the approximation of the instantaneous velocity of the particle’s motion. These two images are required to be still and have no streaks in the particle which is being examined, in order to have minimum results error. In addition, the second way to minimize the error is to shorten the interframe time. With this, a high-speed camera is needed to capture a short interframe image time difference, and to also prepare such as the double pulsed Nd-YAG laser to give a sufficient lighting frequency, might cost more and quite bulky to set up.

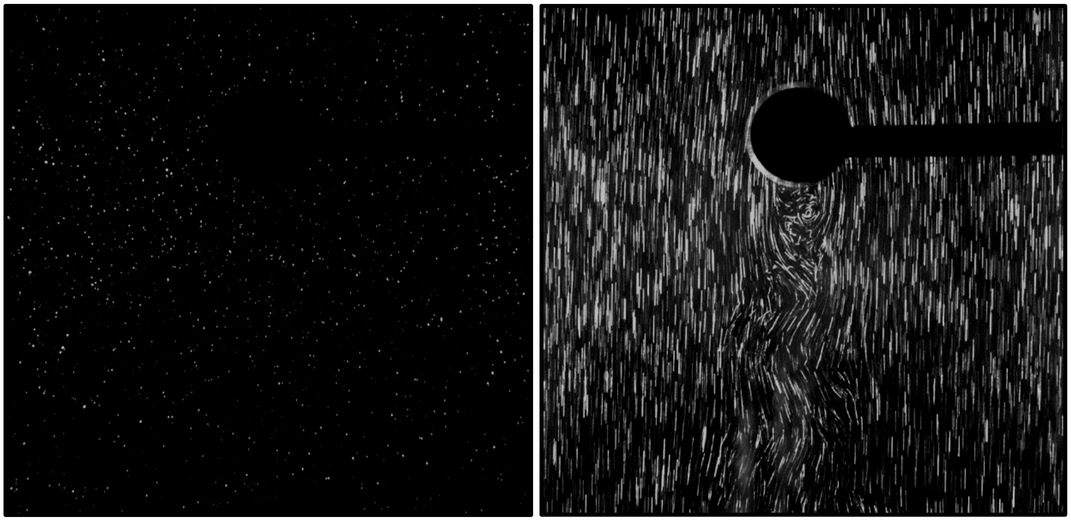


Figure 1.A. The Particle Image (Left image; [1]Short exposure time, [2]Higher hardware requirements) and Streak Image (Right image; [1]Long exposure time, [2]Lower cost for equipment) comparisons. (Image Source: Performances comparison of particle tracking

velocimetry (PTV) and particle image velocimetry (PIV) with long exposure particle streaks - Mumtaz Hussain QURESHI , Wei-Hsin TIEN, and Yi-Jiun Peter LIN)

These are the problems that we may encounter if we choose to do the experiment with these two techniques, which might be unhandy for those who don't have such high-end equipment. The Particle Streak Velocimetry (PSV), is the technique that is developed to process an image with streaks that may occur due to relatively long-exposure time (especially when the captured particle is moving at a relatively high velocity). This classical PSV research concept originated from streak photography as a fundamental tool for flow visualization. With this technique, more technical boundaries are eliminated.

The tools/apparatus used in the experiment cost less, compared to the other two (PIV and PTV) methods. Various components needed in the experiment are easily available for academic purposes. Even with a standard computer, data processing is still achievable. Although PSV has such benefits, there are two problems we must consider. Firstly is that the resolution of the measurement system is less accurate, and finding corresponding streaks in frames might be very challenging.

In streak photography, we can extract more qualitative and quantitative information for a relatively large-scaled flow, in comparison to the other intrusive methods. And as time goes by, the streak photography method has developed into the "*Digital Image Processing of Particle Streak Photography Technique*", which is utilized to do research on finding the streamwise and perpendicular components of the velocity field.

In previous studies of the PSV method, the velocities are assumed to be constant. This idea creates an error, especially when the streak is curved. From the Lagrangian point of view, the particle's velocity changes tangentially along the curved streak trajectory. Basic polynomial fitting generated streaks are not considered as a function of time, so the streak image cannot be regulated by increasing the integration time interval. With this, the least square curve fitting is the new method that has been studied for the purpose of velocity field measurement by a novel streak-resolving algorithm. The algorithm was proven to be successful and robust, based on the "*A Novel Streak-Resolving Algorithm for Particle Streak Velocimetry* - Mumtaz Hussain QURESHI and Wei-Hsin TIEN" research paper.

Before doing this experiment, there are two limitations that must be acknowledged by the readers. First is that the current research is assumed to be laminar, two-dimensional flow, and secondly is that only in-plane motion of the seeding particles is considered. For turbulent flows, the fluctuated velocity is observed as a mean and if the exposure time is longer than the turbulent time scale, the results from the streak image are not valid even though the velocity can be resolved normally by the least square curve fitting algorithm.

II. Motivation

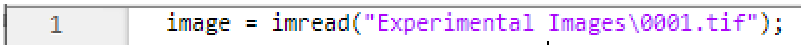
Since the PTV and PIV method might require more advanced hardware that might cost more, it might be a constraint for some people to do the research on this field. So with the newly developed PSV method which doesn't require advanced hardware such as PTV and PIV method, researchers still can resolve the particle properties, and plot velocity vectors with the particle properties which are obtained from the MATLAB-coded algorithm.

III. Description of Research Work

The method involved in doing this research is that you must do an experiment to obtain the experimental images beforehand, and the suitable experimental image would be a streak image of the observed specimen. The suggested image format that is suitable for the MATLAB code is to have the file in 8 until 32-bit depth, and the file dimensions to be in 1392x1040 pixels. This is required because the "imagecropper.m" MATLAB code has been set to process images with these fixed dimensions and properties settings.

A. The imagecropper.m code

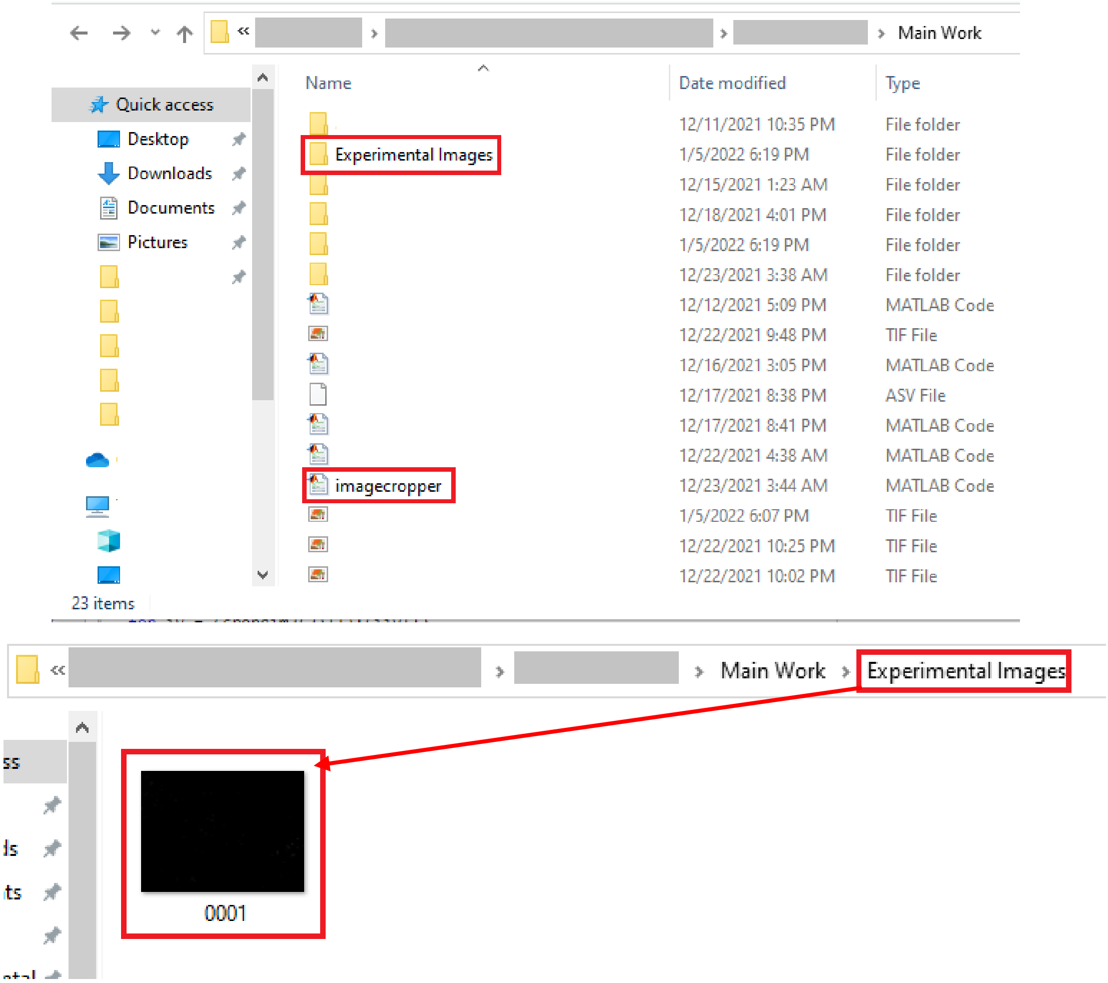
The imagecropper code is an **optional process** if only a small area of streak image from a bigger streak image file is desired to be experimented on. The code is used to crop the image which has a 1392 x 1040 pixels dimension. First of all, an image file is uploaded into the imagecropper.m file. The upload process of the image file can be done by locating the directory of the desired image file, and typing it down in the "imread("*Type the image file directory here*")". Figure **2.A.** below is an example of the currently discussed process :



```
image = imread("Experimental Images\0001.tif");
```

Figure 2.A. Image uploading format.

The figure shown above informs us that in the folder that the imagecropper.m code is located, there also exists a folder named "**Experimental Images**". This folder contains the image file that we want to process in the imagecropper.m code. The image file that we are going to upload is the "**0001.tif**" file. (refer to **Figure 2.B** for the file directory image)



« [redacted] › [redacted] › [redacted] › Main Work

Name	Date modified	Type
	12/11/2021 10:35 PM	File folder
Experimental Images	1/5/2022 6:19 PM	File folder
	12/15/2021 1:23 AM	File folder
	12/18/2021 4:01 PM	File folder
	1/5/2022 6:19 PM	File folder
	12/23/2021 3:38 AM	File folder
	12/12/2021 5:09 PM	MATLAB Code
	12/22/2021 9:48 PM	TIF File
	12/16/2021 3:05 PM	MATLAB Code
	12/17/2021 8:38 PM	ASV File
	12/17/2021 8:41 PM	MATLAB Code
	12/22/2021 4:38 AM	MATLAB Code
imagecropper	12/23/2021 3:44 AM	MATLAB Code
	1/5/2022 6:07 PM	TIF File
	12/22/2021 10:25 PM	TIF File
	12/22/2021 10:02 PM	TIF File

« [redacted] › [redacted] › Main Work › Experimental Images

Figure 2.B Image file directory.

After inputting the image that we want to process, the code will show a figure, that allows us to crop the streak area manually. The following tab, shown in **Figure 2.C.** will demonstrate what is going to pop-up right after running the code.

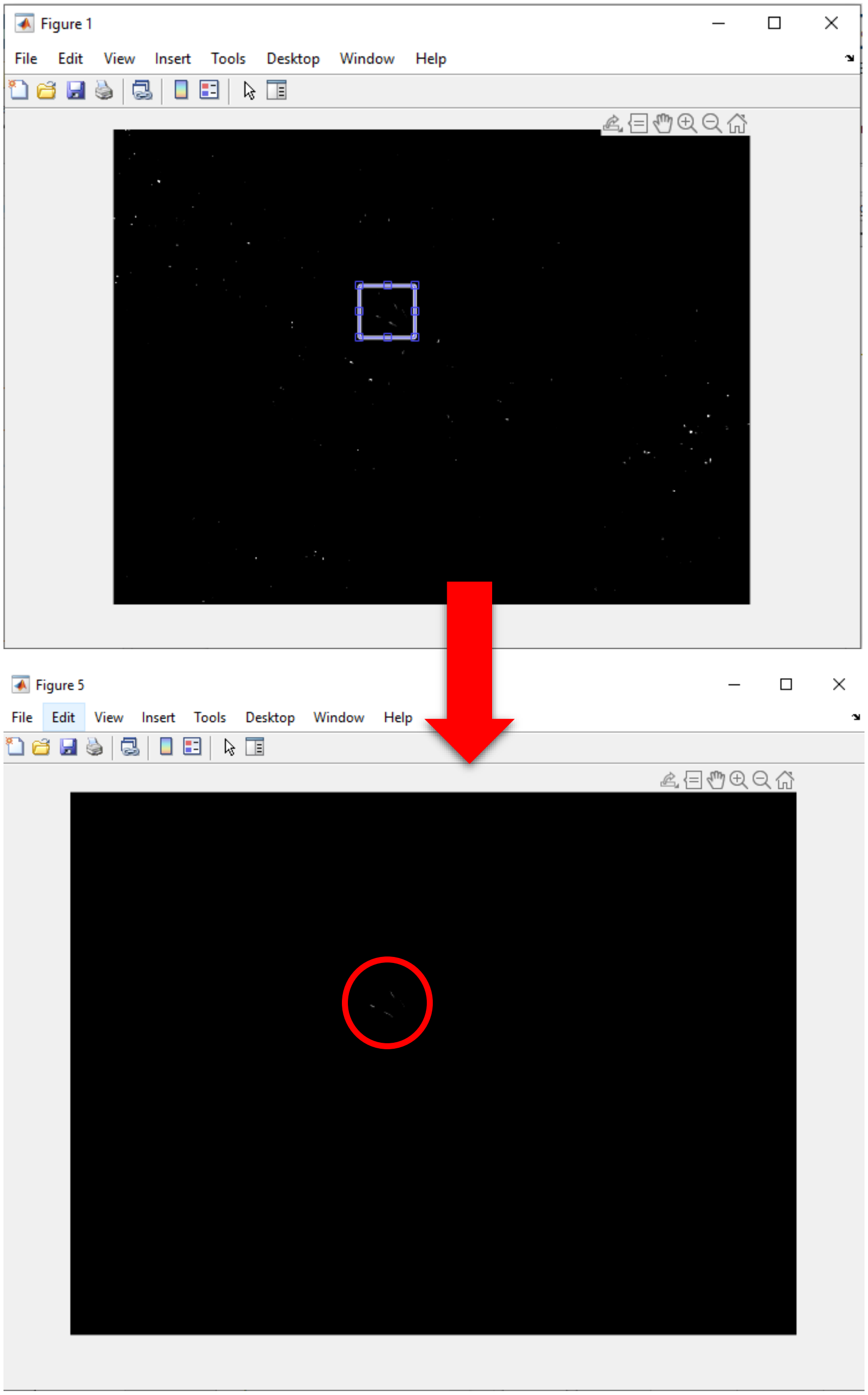
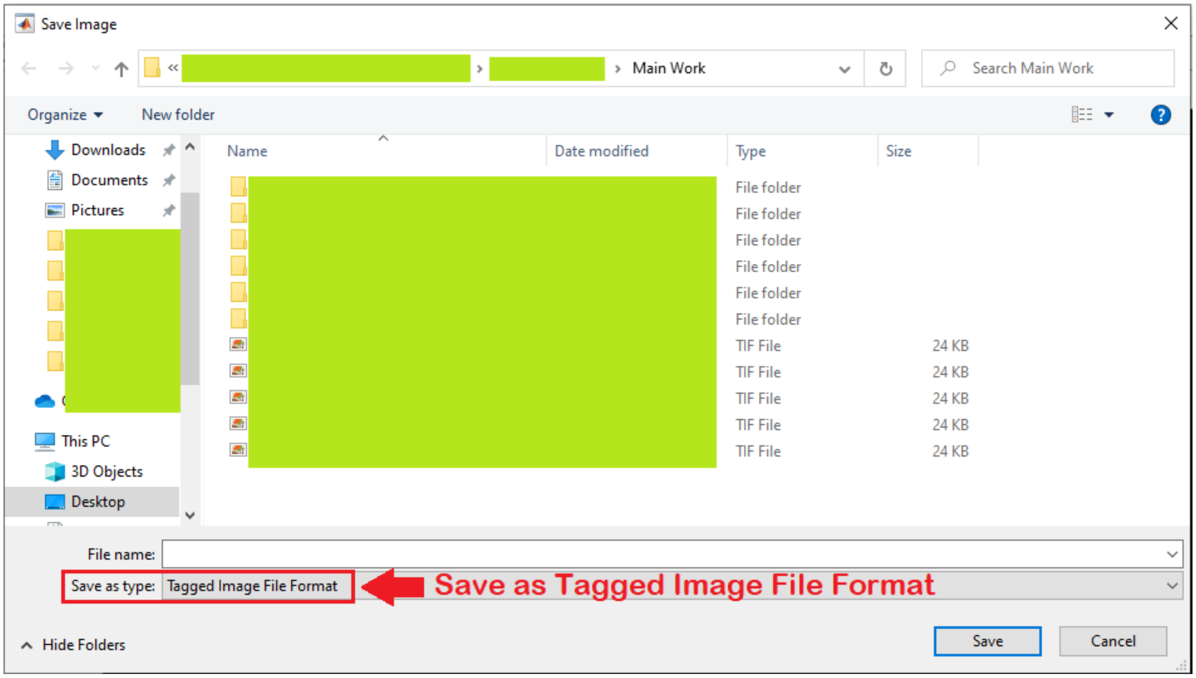


Figure 1
File Edit View Insert Tools Desktop Window Help
Figure 5
File Edit View Insert Tools Desktop Window Help

Figure 2.C. Image Cropping tab (After the crop selection box, we can see the newly generated image file which contains only the streaks that were chosen marked with the red circle).

After the desired streak image area has been selected, double click the selection box and the code will generate a new image that only contains the selected streaks in the area, it is suggested to make sure that none of the streaks are cropped/cut so that the iteration can achieve the desired results, and to prohibit any further errors and miscalculations. After double-clicking the selection box, a new save file tab will appear. It is shown in **Figure 2.D.**below.



Name	Date modified	Type	Size
[redacted]	[redacted]	File folder	
[redacted]	[redacted]	File folder	
[redacted]	[redacted]	File folder	
[redacted]	[redacted]	File folder	
[redacted]	[redacted]	File folder	
[redacted]	[redacted]	File folder	
[redacted]	[redacted]	TIF File	24 KB
[redacted]	[redacted]	TIF File	24 KB
[redacted]	[redacted]	TIF File	24 KB
[redacted]	[redacted]	TIF File	24 KB
[redacted]	[redacted]	TIF File	24 KB

File name:

Save as type: Tagged Image File Format

Save as Tagged Image File Format

Figure 2.D. Save Image Tab.

When the file save tab appears, it is recommended to save the file into the original image file’s format. In the experimental images that are used, the format of the image is .TIF, and therefore, the image outputted by this code will have .TIF format as well.

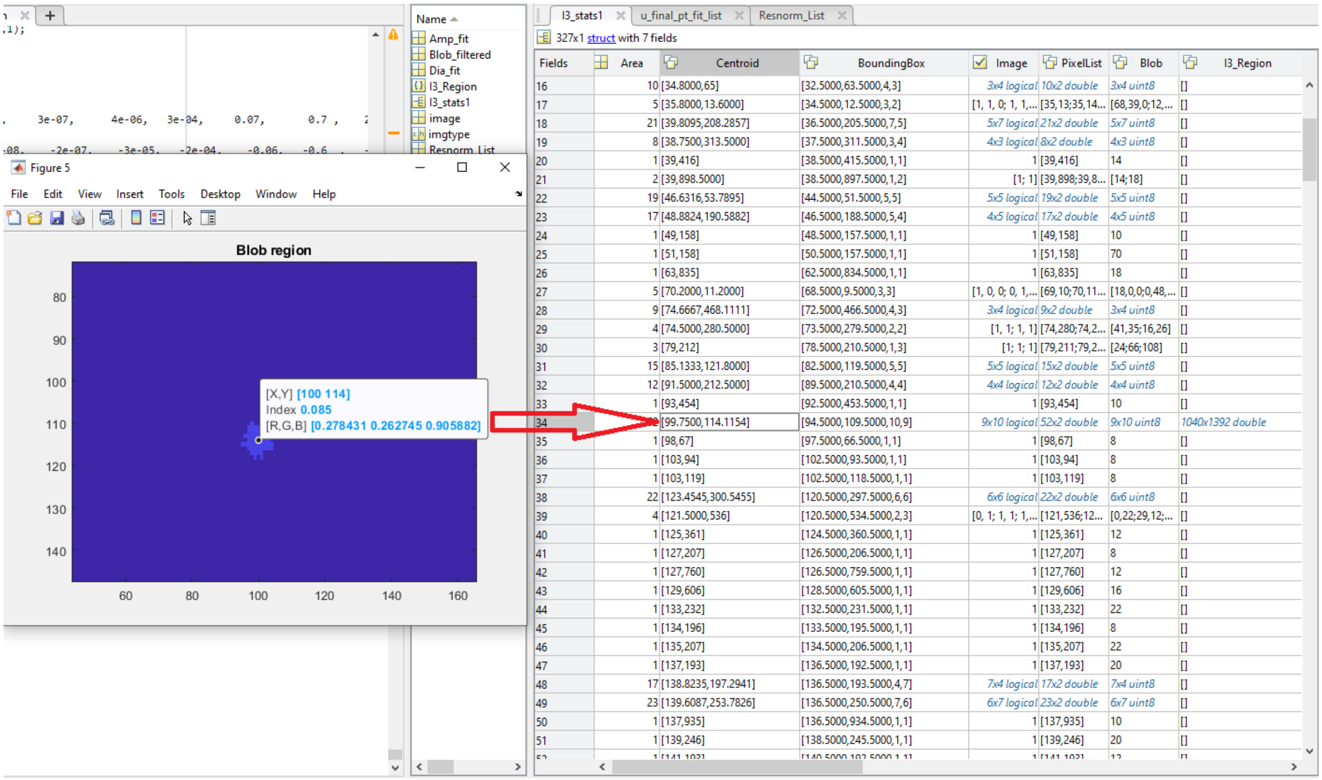
As discussed earlier, there is a reason why we suggest the user input a 1392 x 1040 dimensioned image. The imagecropper.m code is designed to crop an area of an image and is used to paste it into another blank canvas. This blank canvas is made of zero matrices with a 1392 x 1040 configuration. The cropped image is then converted as a matrix and then pasted to the pre-prepared blank 1392 x 1040 canvas.

B. The streakprocess.m code

The second code is designed for resolving the input file's streak image. The algorithm of the second code is developed by Mumtaz Hussain Qureshi. The streak processing algorithm requires an input of 8,16 or 32-bit images. The importance of recognizing the file's bit-depth is to keep the intensity between 0 and 1. For 8-bit images, we divide the original intensity by 255; and for 16 bits by 65535; and finally for 32 bits it is divided by 4294967295. The following image is the streak image that is used for demonstrating the whole code (refer to the **Figure 2.E.** below).

Figure 2.E. Experimental image preview (.TIF file, 8-bits depth, 1392 x 1040 pixels; shown in **Appendix** page 34).

When the streak processing code is invoked in a coding line, it requires the streak image file input and it directly dissects each streak into different areas. It groups each different streak by area, and count for how many streak(s) were present in the image. It registers each area's bounding box, which tells the reader from which pixel to which pixel does the streak cover (the coordinate starts from the top-left to bottom-right of the streak's area). Centroid's coordinates are also provided so that we may locate the corresponding easily. Other than these two, there are extra information such as Image binarized intensity profile; pixels list (which pixels are covered by the streak, coordinates in [X, Y]); the blob region intensity profile; and finally, the I3_region which shows us which streaks are eligible for velocity component calculations, and also the intensity matrix of the corresponding blob area.



327x1 struct with 7 fields

Fields	Area	Centroid	BoundingBox	Image	PixelList	Blob	I3_Region
16	10	[34.8000,65]	[32.5000,63.5000,4,3]	3x4 logical	10x2 double	3x4 uint8	[]
17	5	[35.8000,13.6000]	[34.5000,12.5000,3,2]	[1, 1, 0; 1, 1,...	[35,13;35,14...	[68,39,0;12,...	[]
18	21	[39.8095,208.2857]	[36.5000,205.5000,7,5]	5x7 logical	21x2 double	5x7 uint8	[]
19	8	[38.7500,313.5000]	[37.5000,311.5000,3,4]	4x3 logical	8x2 double	4x3 uint8	[]
20	1	[39,416]	[38.5000,415.5000,1,1]	1	[39,416]	14	[]
21	2	[39,898.5000]	[38.5000,897.5000,1,2]	[1; 1]	[39,898;39,8...	[14;18]	[]
22	19	[46.6316,53.7895]	[44.5000,51.5000,5,5]	5x5 logical	19x2 double	5x5 uint8	[]
23	17	[48.8824,190.5882]	[46.5000,188.5000,5,4]	4x5 logical	17x2 double	4x5 uint8	[]
24	1	[49,158]	[48.5000,157.5000,1,1]	1	[49,158]	10	[]
25	1	[51,158]	[50.5000,157.5000,1,1]	1	[51,158]	70	[]
26	1	[63,835]	[62.5000,834.5000,1,1]	1	[63,835]	18	[]
27	5	[70.2000,11.2000]	[68.5000,9.5000,3,3]	[1, 0, 0; 0, 1,...	[69,10;70,11...	[18,0,0;0,48,...	[]
28	9	[74.6667,468.1111]	[72.5000,466.5000,4,3]	3x4 logical	9x2 double	3x4 uint8	[]
29	4	[74.5000,280.5000]	[73.5000,279.5000,2,2]	[1, 1; 1, 1]	[74,280;74,2...	[41,35;16,26]	[]
30	3	[79,212]	[78.5000,210.5000,1,3]	[1; 1; 1]	[79,211;79,2...	[24;66;108]	[]
31	15	[85.1333,121.8000]	[82.5000,119.5000,5,5]	5x5 logical	15x2 double	5x5 uint8	[]
32	12	[91.5000,212.5000]	[89.5000,210.5000,4,4]	4x4 logical	12x2 double	4x4 uint8	[]
33	1	[93,454]	[92.5000,453.5000,1,1]	1	[93,454]	10	[]
34	[illegible]	[99.7500,114.1154]	[94.5000,109.5000,10,9]	9x10 logical	52x2 double	9x10 uint8	1040x1392 double
35	1	[98,67]	[97.5000,66.5000,1,1]	1	[98,67]	8	[]
36	1	[103,94]	[102.5000,93.5000,1,1]	1	[103,94]	8	[]
37	1	[103,119]	[102.5000,118.5000,1,1]	1	[103,119]	8	[]
38	22	[123.4545,300.5455]	[120.5000,297.5000,6,6]	6x6 logical	22x2 double	6x6 uint8	[]
39	4	[121.5000,536]	[120.5000,534.5000,2,3]	[0, 1; 1, 1; 1,...	[121,536;12...	[0,22;29,12;...	[]
40	1	[125,361]	[124.5000,360.5000,1,1]	1	[125,361]	12	[]
41	1	[127,207]	[126.5000,206.5000,1,1]	1	[127,207]	8	[]
42	1	[127,760]	[126.5000,759.5000,1,1]	1	[127,760]	12	[]
43	1	[129,606]	[128.5000,605.5000,1,1]	1	[129,606]	16	[]
44	1	[133,232]	[132.5000,231.5000,1,1]	1	[133,232]	22	[]
45	1	[134,196]	[133.5000,195.5000,1,1]	1	[134,196]	8	[]
46	1	[135,207]	[134.5000,206.5000,1,1]	1	[135,207]	22	[]
47	1	[137,193]	[136.5000,192.5000,1,1]	1	[137,193]	20	[]
48	17	[138.8235,197.2941]	[136.5000,193.5000,4,7]	7x4 logical	17x2 double	7x4 uint8	[]
49	23	[139.6087,253.7826]	[136.5000,250.5000,7,6]	6x7 logical	23x2 double	6x7 uint8	[]
50	1	[137,935]	[136.5000,934.5000,1,1]	1	[137,935]	10	[]
51	1	[139,246]	[138.5000,245.5000,1,1]	1	[139,246]	20	[]

Figure 2.F. Finding the streak coordinates in the figure by utilizing the centroid's location.

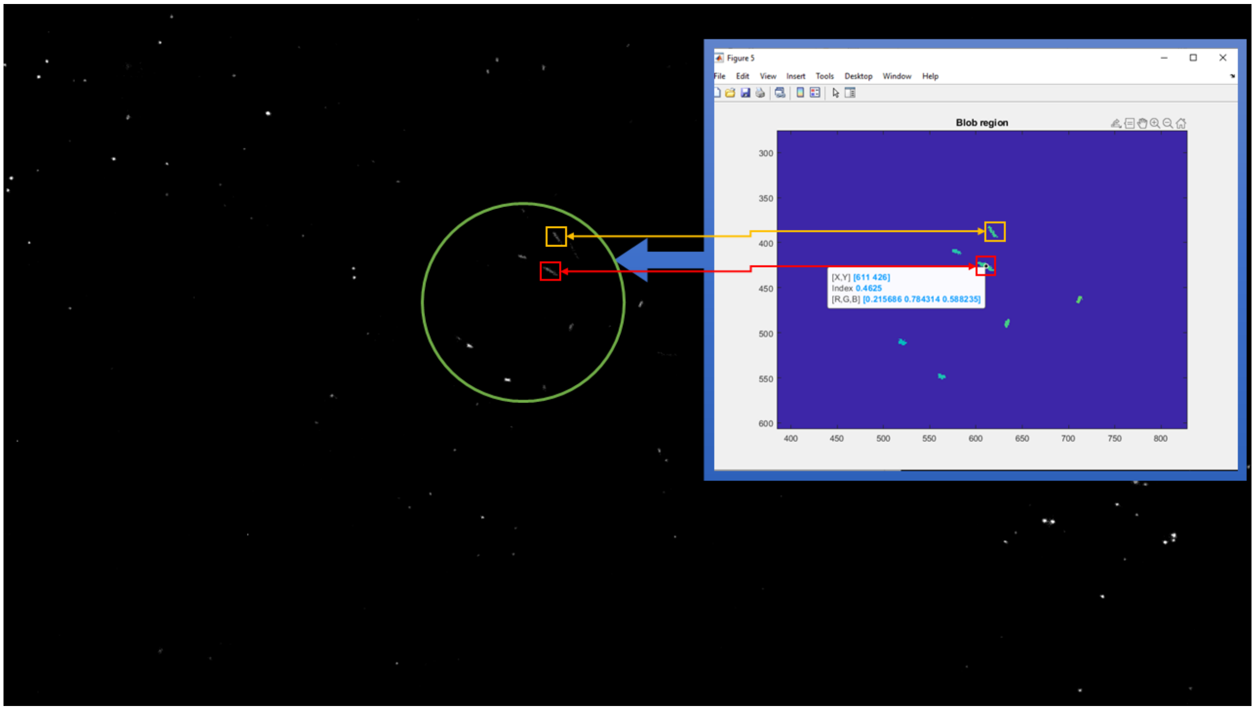
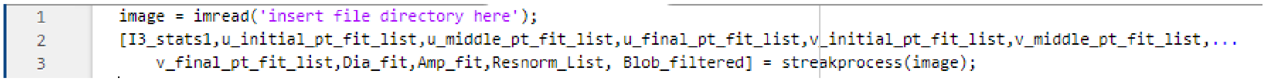


Figure 2.G. The conversion from the original image to the blob region figures, only blob areas with an area above 30 pixels are inputted into the figure.

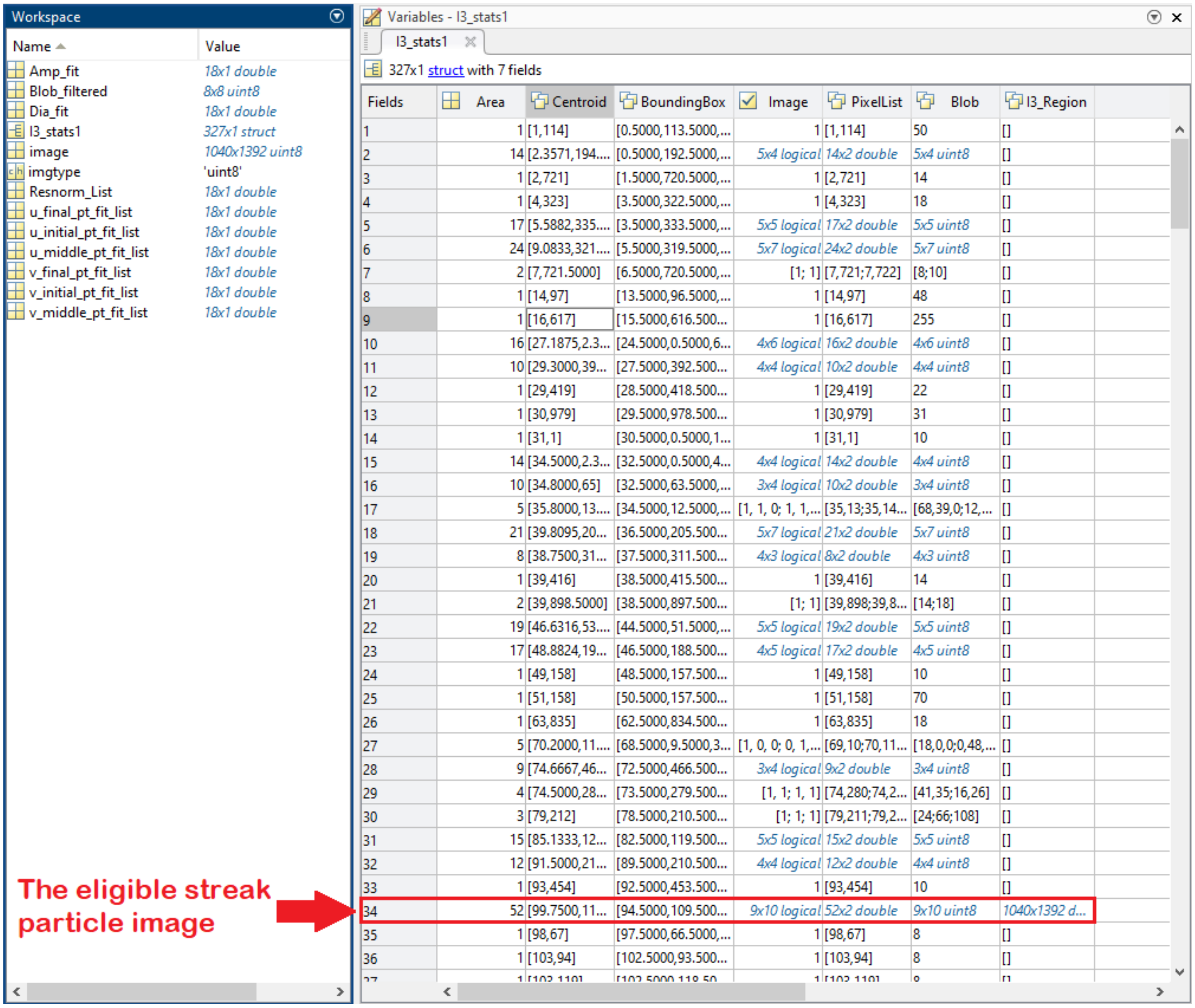
The streakprocess code is invoked in a code line by just typing in the "streakprocess(image_file)" and it will jump into its code iteration. Depending on how many streaks are present in the image, the code run-time varies. The more streaks that exist in an image file, the longer time it will need to process all of the streak properties. Before invoking the streakprocess code, make sure that the streakprocess.m file is opened. **Figure 2.H.** below will demonstrate for how to invoke the streakprocess code.



```
image = imread('insert file directory here');
[I3_stats1,u_initial_pt_fit_list,u_middle_pt_fit_list,u_final_pt_fit_list,v_initial_pt_fit_list,v_middle_pt_fit_list,...
    v_final_pt_fit_list,Dia_fit,Amp_fit,Resnorm_List, Blob_filtered] = streakprocess(image);
```

Figure 2.H. Example on how to call the streakprocess function.

The streak processing algorithm detects all of the existing streaks, including the relatively small area streak blobs. But the limitations that we use in this algorithm is that a streak's velocity properties will not be calculated if the area is less than 30 pixels (the area limitation depends on the desired requirements, and this experiment will use a minimum area of 30 pixels). This might increase the efficiency of the iteration time because the algorithm only focuses on the major/bigger streaks. Such simplification needs to be done because the iteration time does take some time, and this might be beneficial for time's sake.



Name	Value
Amp_fit	18x1 double
Blob_filtered	8x8 uint8
Dia_fit	18x1 double
I3_stats1	327x1 struct
image	1040x1392 uint8
imgtype	'uint8'
Resnorm_List	18x1 double
u_final_pt_fit_list	18x1 double
u_initial_pt_fit_list	18x1 double
u_middle_pt_fit_list	18x1 double
v_final_pt_fit_list	18x1 double
v_initial_pt_fit_list	18x1 double
v_middle_pt_fit_list	18x1 double

327x1 struct with 7 fields

Fields	Area	Centroid	BoundingBox	Image	PixelList	Blob	I3_Region
1	1	[1,114]	[0.5000,113.5000,...	1	[1,114]	50	[]
2	14	[2.3571,194....	[0.5000,192.5000,...	5x4 logical	14x2 double	5x4 uint8	[]
3	1	[2,721]	[1.5000,720.5000,...	1	[2,721]	14	[]
4	1	[4,323]	[3.5000,322.5000,...	1	[4,323]	18	[]
5	17	[5.5882,335....	[3.5000,333.5000,...	5x5 logical	17x2 double	5x5 uint8	[]
6	24	[9.0833,321....	[5.5000,319.5000,...	5x7 logical	24x2 double	5x7 uint8	[]
7	2	[7,721.5000]	[6.5000,720.5000,...	[1; 1]	[7,721;7,722]	[8;10]	[]
8	1	[14,97]	[13.5000,96.5000,...	1	[14,97]	48	[]
9	1	[16,617]	[15.5000,616.500...	1	[16,617]	255	[]
10	16	[27.1875,2.3...	[24.5000,0.5000,6...	4x6 logical	16x2 double	4x6 uint8	[]
11	10	[29.3000,39...	[27.5000,392.500...	4x4 logical	10x2 double	4x4 uint8	[]
12	1	[29,419]	[28.5000,418.500...	1	[29,419]	22	[]
13	1	[30,979]	[29.5000,978.500...	1	[30,979]	31	[]
14	1	[31,1]	[30.5000,0.5000,1...	1	[31,1]	10	[]
15	14	[34.5000,2.3...	[32.5000,0.5000,4...	4x4 logical	14x2 double	4x4 uint8	[]
16	10	[34.8000,65]	[32.5000,63.5000,...	3x4 logical	10x2 double	3x4 uint8	[]
17	5	[35.8000,13....	[34.5000,12.5000,...	[1, 1, 0; 1, 1,...	[35,13;35,14...	[68,39,0;12,...	[]
18	21	[39.8095,20...	[36.5000,205.500...	5x7 logical	21x2 double	5x7 uint8	[]
19	8	[38.7500,31...	[37.5000,311.500...	4x3 logical	8x2 double	4x3 uint8	[]
20	1	[39,416]	[38.5000,415.500...	1	[39,416]	14	[]
21	2	[39,898.5000]	[38.5000,897.500...	[1; 1]	[39,898;39,8...	[14;18]	[]
22	19	[46.6316,53....	[44.5000,51.5000,...	5x5 logical	19x2 double	5x5 uint8	[]
23	17	[48.8824,19...	[46.5000,188.500...	4x5 logical	17x2 double	4x5 uint8	[]
24	1	[49,158]	[48.5000,157.500...	1	[49,158]	10	[]
25	1	[51,158]	[50.5000,157.500...	1	[51,158]	70	[]
26	1	[63,835]	[62.5000,834.500...	1	[63,835]	18	[]
27	5	[70.2000,11....	[68.5000,9.5000,3...	[1, 0, 0; 0, 1,...	[69,10;70,11...	[18,0,0;0,48,...	[]
28	9	[74.6667,46...	[72.5000,466.500...	3x4 logical	9x2 double	3x4 uint8	[]
29	4	[74.5000,28...	[73.5000,279.500...	[1, 1; 1, 1]	[74,280;74,2...	[41,35;16,26]	[]
30	3	[79,212]	[78.5000,210.500...	[1; 1; 1]	[79,211;79,2...	[24;66;108]	[]
31	15	[85.1333,12...	[82.5000,119.500...	5x5 logical	15x2 double	5x5 uint8	[]
32	12	[91.5000,21...	[89.5000,210.500...	4x4 logical	12x2 double	4x4 uint8	[]
33	1	[93,454]	[92.5000,453.500...	1	[93,454]	10	[]
34	52	[99.7500,11...	[94.5000,109.500...	9x10 logical	52x2 double	9x10 uint8	1040x1392 d...
35	1	[98,67]	[97.5000,66.5000,...	1	[98,67]	8	[]
36	1	[103,94]	[102.5000,93.500...	1	[103,94]	8	[]

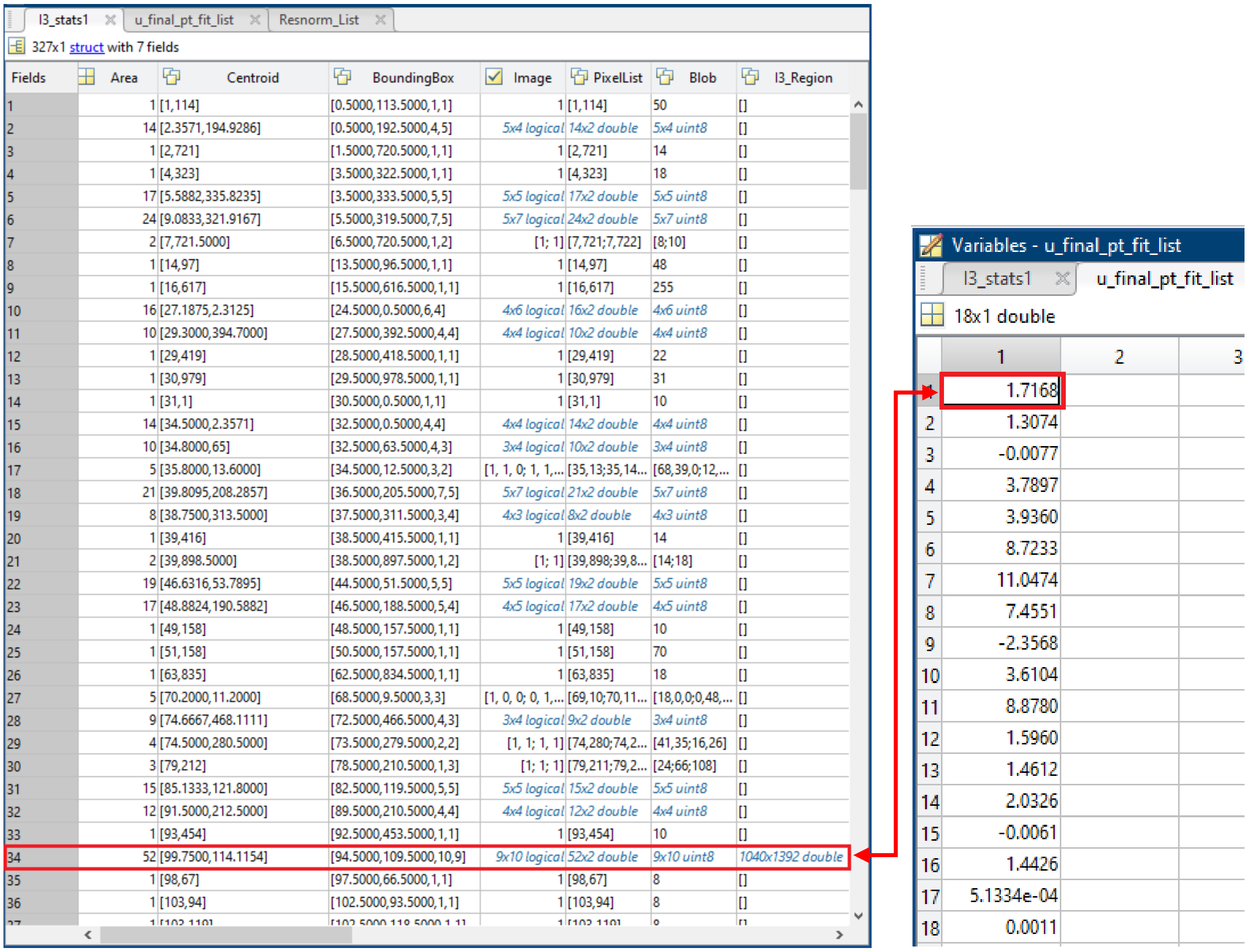
Figure 2.I. The eligible/filtered particle streak image is shown in the workspace after the code finished resolving the streak images.

As we can see from **Figure 2.I.** above, the filtered streak particles are the ones that have an area above 30 pixels. This criterion can be changed based on the experiment requirements, but for this example, the criterion used is that the minimum pixel area (thr_area = 30; streakprocess.m code line 65) is set to be 30 pixels. Streak particle number 34 has an area of 52 pixels, therefore is eligible for velocity properties calculation. The I3_Region column shows us for which particles are eligible based on the blob area, and they are marked with the dimension of the image file i.e., "1040x1392 double" instead of

“[]”. The “[]” in the I3_region column means that the corresponding particle streak velocity components are not going to be calculated because the area is too small.

The next part is the least-square curve fitting iteration. The particle streak’s properties are then calculated and resolved with the least-square curve fitting algorithm. This multi-variable least-square curve-fitting is done by utilizing the lsqcurvefit function in MATLAB’s optimization toolbox. The first perimeter called “xdata” is the list of the pixel’s locations as the input array, cropped from the real experimental images. Secondly is the “ydata”, which is the observed output array and it represents the pixel intensity values at each pixel location shown by the “xdata”. The least-square curve fitting algorithm’s performance is shown as the residual values (squared 2-norm of the residual). Low residual values indicated that the resolved particle streak image is close to the true streak values and close to the true trajectory.

Finally, the thing that can be seen from the Workspace area is that “I3_stats” is consisted of a 327 x 1 structure. Besides the “I3_stats” variable, most of the other variables indicate that they have a data file of 18 x 1 double. This means that the code detected 327 different streaks, but the ones that managed to be filtered with respect to the area size are only 18 streaks. These 18 streaks’ velocity components are then calculated and their data are displayed in each different variable that exists in the workspace tab. To correlate for which streak owns which velocity components, the nth-streak component in the table (i.e., u_initial _pt_fit/ properties other than I3_stats) is u initial point velocity component of the nth-streak in the “I3_stats” table which has the “I3_Region” of the image’s size matrix (i.e., “1040 x 1392 double”, not the “[]” ones (figure explanation in **Figure 2.J.**)).



327x1 struct with 7 fields

Fields	Area	Centroid	BoundingBox	Image	PixelList	Blob	I3_Region
1	1	[1,114]	[0.5000,113.5000,1,1]	1	[1,114]	50	[]
2	14	[2.3571,194.9286]	[0.5000,192.5000,4,5]	5x4 logical	14x2 double	5x4 uint8	[]
3	1	[2,721]	[1.5000,720.5000,1,1]	1	[2,721]	14	[]
4	1	[4,323]	[3.5000,322.5000,1,1]	1	[4,323]	18	[]
5	17	[5.5882,335.8235]	[3.5000,333.5000,5,5]	5x5 logical	17x2 double	5x5 uint8	[]
6	24	[9.0833,321.9167]	[5.5000,319.5000,7,5]	5x7 logical	24x2 double	5x7 uint8	[]
7	2	[7,721.5000]	[6.5000,720.5000,1,2]	[1; 1]	[7,721;7,722]	[8;10]	[]
8	1	[14,97]	[13.5000,96.5000,1,1]	1	[14,97]	48	[]
9	1	[16,617]	[15.5000,616.5000,1,1]	1	[16,617]	255	[]
10	16	[27.1875,2.3125]	[24.5000,0.5000,6,4]	4x6 logical	16x2 double	4x6 uint8	[]
11	10	[29.3000,394.7000]	[27.5000,392.5000,4,4]	4x4 logical	10x2 double	4x4 uint8	[]
12	1	[29,419]	[28.5000,418.5000,1,1]	1	[29,419]	22	[]
13	1	[30,979]	[29.5000,978.5000,1,1]	1	[30,979]	31	[]
14	1	[31,1]	[30.5000,0.5000,1,1]	1	[31,1]	10	[]
15	14	[34.5000,2.3571]	[32.5000,0.5000,4,4]	4x4 logical	14x2 double	4x4 uint8	[]
16	10	[34.8000,65]	[32.5000,63.5000,4,3]	3x4 logical	10x2 double	3x4 uint8	[]
17	5	[35.8000,13.6000]	[34.5000,12.5000,3,2]	[1, 1, 0; 1, 1,...	[35,13;35,14...	[68,39,0;12,...	[]
18	21	[39.8095,208.2857]	[36.5000,205.5000,7,5]	5x7 logical	21x2 double	5x7 uint8	[]
19	8	[38.7500,313.5000]	[37.5000,311.5000,3,4]	4x3 logical	8x2 double	4x3 uint8	[]
20	1	[39,416]	[38.5000,415.5000,1,1]	1	[39,416]	14	[]
21	2	[39,898.5000]	[38.5000,897.5000,1,2]	[1; 1]	[39,898;39,8...	[14;18]	[]
22	19	[46.6316,53.7895]	[44.5000,51.5000,5,5]	5x5 logical	19x2 double	5x5 uint8	[]
23	17	[48.8824,190.5882]	[46.5000,188.5000,5,4]	4x5 logical	17x2 double	4x5 uint8	[]
24	1	[49,158]	[48.5000,157.5000,1,1]	1	[49,158]	10	[]
25	1	[51,158]	[50.5000,157.5000,1,1]	1	[51,158]	70	[]
26	1	[63,835]	[62.5000,834.5000,1,1]	1	[63,835]	18	[]
27	5	[70.2000,11.2000]	[68.5000,9.5000,3,3]	[1, 0, 0; 0, 1,...	[69,10;70,11...	[18,0,0;0,48,...	[]
28	9	[74.6667,468.1111]	[72.5000,466.5000,4,3]	3x4 logical	9x2 double	3x4 uint8	[]
29	4	[74.5000,280.5000]	[73.5000,279.5000,2,2]	[1, 1; 1, 1]	[74,280;74,2...	[41,35;16,26]	[]
30	3	[79,212]	[78.5000,210.5000,1,3]	[1; 1; 1]	[79,211;79,2...	[24;66;108]	[]
31	15	[85.1333,121.8000]	[82.5000,119.5000,5,5]	5x5 logical	15x2 double	5x5 uint8	[]
32	12	[91.5000,212.5000]	[89.5000,210.5000,4,4]	4x4 logical	12x2 double	4x4 uint8	[]
33	1	[93,454]	[92.5000,453.5000,1,1]	1	[93,454]	10	[]
34	52	[99.7500,114.1154]	[94.5000,109.5000,10,9]	9x10 logical	52x2 double	9x10 uint8	1040x1392 double
35	1	[98,67]	[97.5000,66.5000,1,1]	1	[98,67]	8	[]
36	1	[103,94]	[102.5000,93.5000,1,1]	1	[103,94]	8	[]

18x1 double

	1	2	3
1	1.7168		
2	1.3074		
3	-0.0077		
4	3.7897		
5	3.9360		
6	8.7233		
7	11.0474		
8	7.4551		
9	-2.3568		
10	3.6104		
11	8.8780		
12	1.5960		
13	1.4612		
14	2.0326		
15	-0.0061		
16	1.4426		
17	5.1334e-04		
18	0.0011		

Figure 2.J. The 1st I3_Region filtered streak is also the 1st row calculated properties.

The finished iteration results are then stored in the workspace, and some properties that are calculated from the iteration processes are :

1. Fitter Amplitude
2. Fitted diameter
3. I3_Region
4. I3_stats1
5. Image
6. Image type (8/16/32 bits)
7. Resnorm/squared 2-norm of the residual list
8. Initial-point U velocity vector
9. Mid-point U velocity vector
10. Final-point U velocity vector

11. Initial-point V velocity vector
12. Mid-point V velocity vector
13. Final-point V velocity vector

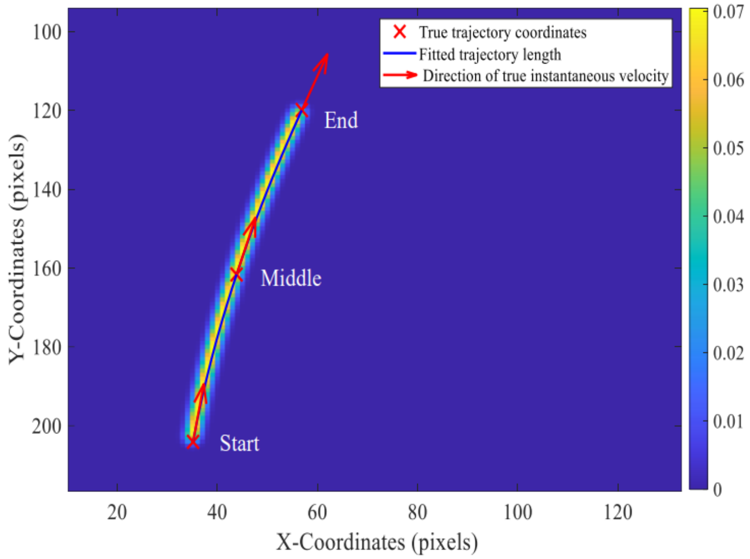


Figure 2.K. The Initial, Mid, and End-point of the streak velocity vector. [*This figure is an example to develop the reader's understanding of the initial, middle, and final point of the velocity vector trajectory from the particle streak's image.]*

C. The vectorplot.m code

After the streakprocess code process the image and obtains the numerical results, we can save each variable result individually. The purpose of saving each variable individually is so that the third code, which is the vectorplot code can load the individual file results and plot the vectors based on the results of the streakprocess code. The variables should be saved in the format of ".mat" (MATLAB data) .

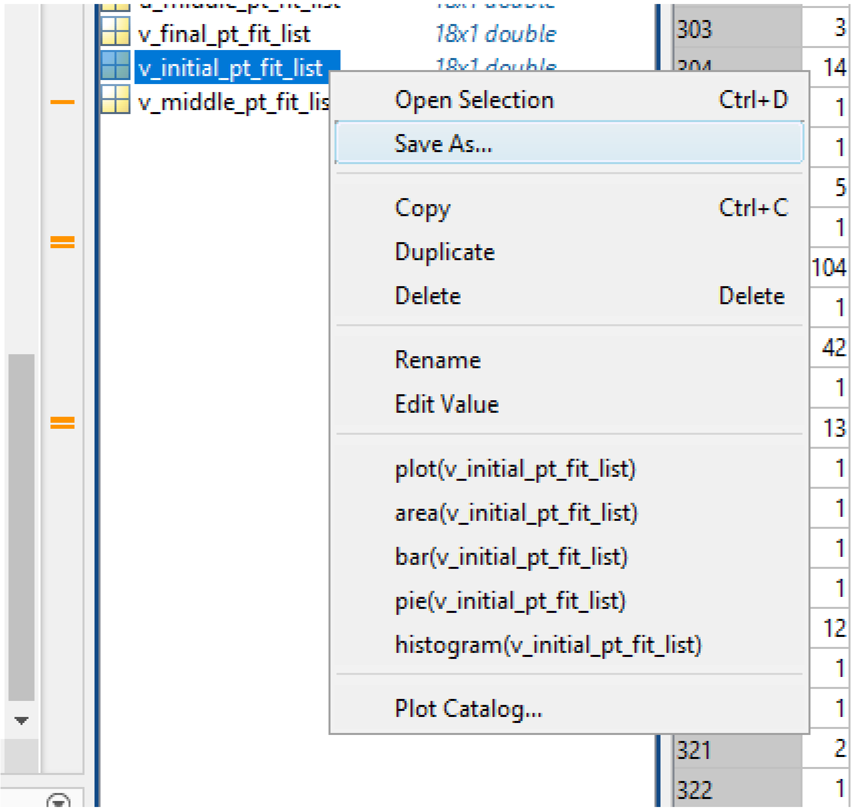
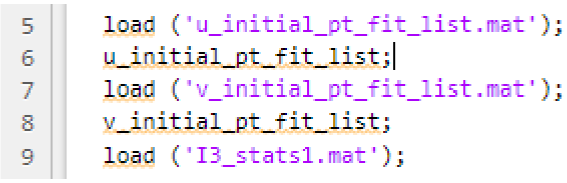


Figure 2.L. Saving a(n) single/individual workspace variable to file directory.

The four variables from the resulting workspace that are needed in the vectorplot code are :

1) u_initial_pt_fit_list
2) v_initial_pt_fit_list
3) I3_stats1
4) Resnorm_List

After saving these 4 variables, the directory of the file must be described in the code's load function to properly load the data(s). The figure below shows us how to call the variables if they are located in the same folder with the vectorplot code.



```
load ('u_initial_pt_fit_list.mat');
u_initial_pt_fit_list;
load ('v_initial_pt_fit_list.mat');
v_initial_pt_fit_list;
load ('I3_stats1.mat');
```

Figure 2.M. The load('*file directory*') syntax loading the file in the chosen directory.

Once these 4 variables are loaded into the code, the code will run its algorithm to calculate the vector trajectories of each streak particle. Then, the result of the vectorplot code will be these 2 figures below:

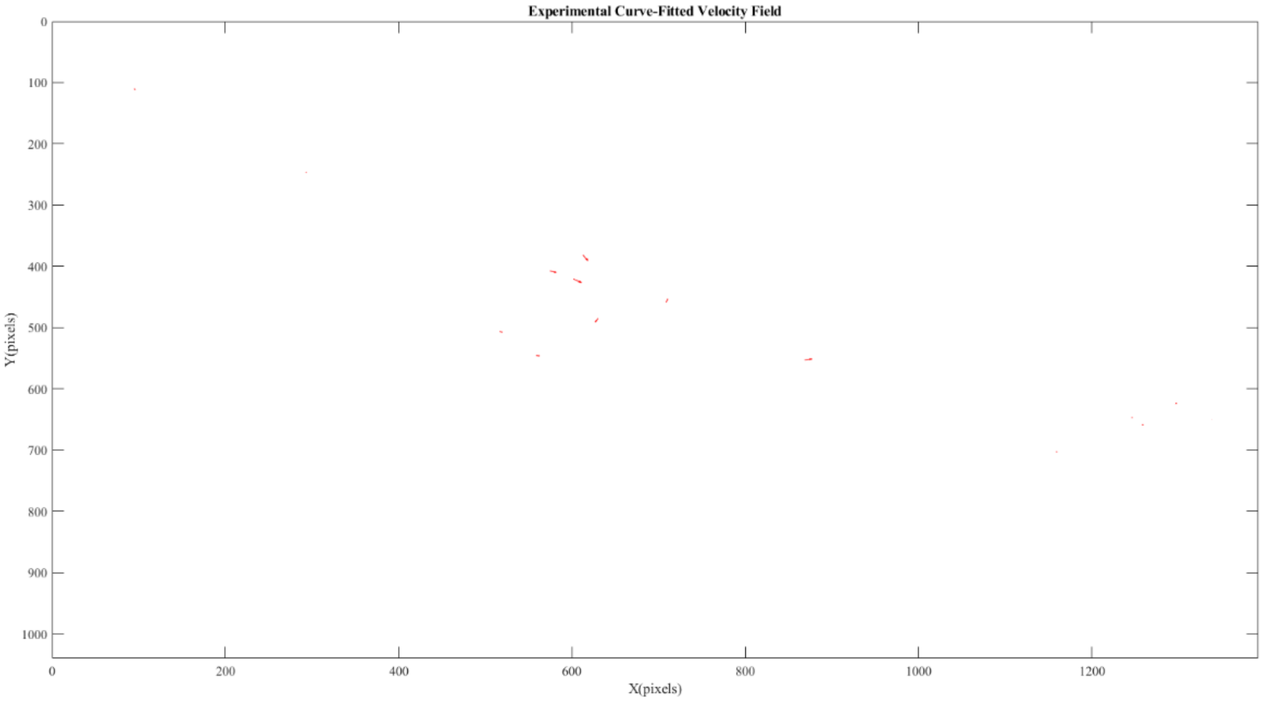


Figure 2.N. Experimental Curve-Fitted Velocity Field vector plot (Shown in **Appendix**, page 35-36).

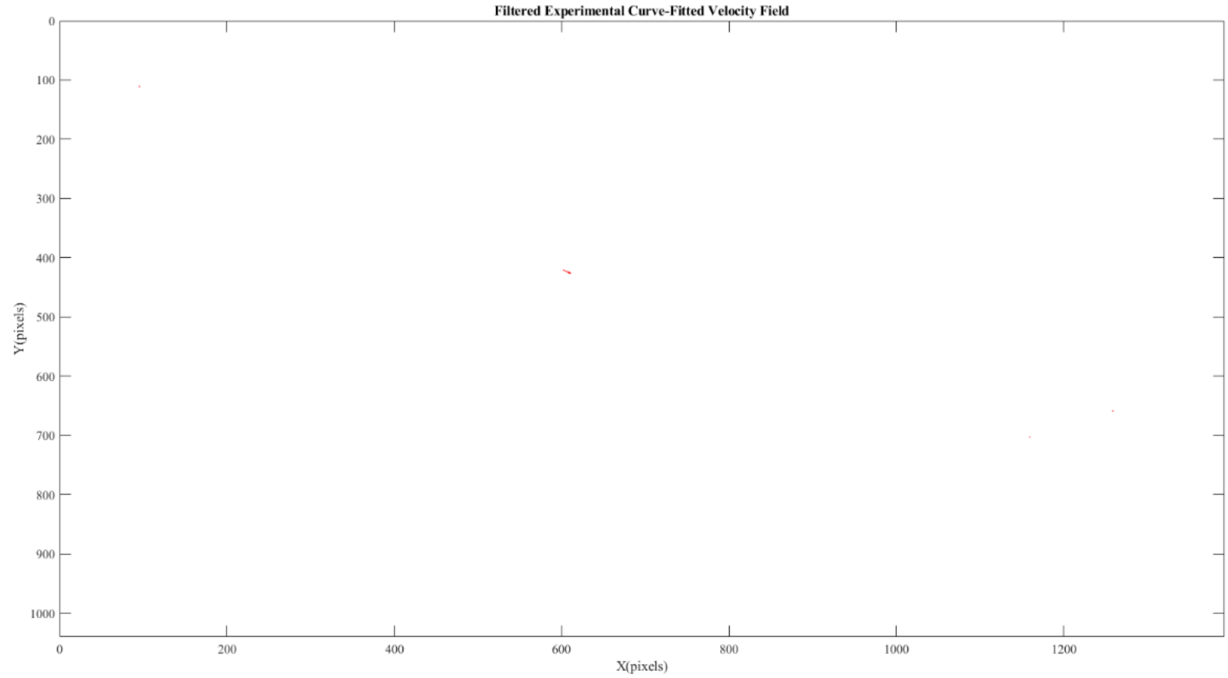
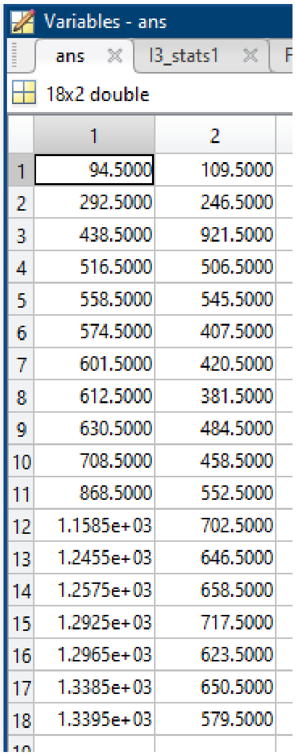


Figure 2.O. Filtered Experimental Curve-Fitted Velocity Field vector plot (shown in **Appendix**, page 37-38).

As the two figures above are not really clear to show, the figures will be shown in the **appendix** part of this paper. The first figure shows us that there are 18 plotted velocity vectors: some are relatively big, and some are relatively too small to be seen. The bigger vector arrow shows that the corresponding streak has a relatively large velocity, compared to the smaller ones. The result in the workspace is a variable named "ans", with a variable type of double, which contains the coordinates for each streak plotted in the "Experimental Curve-Fitted Velocity Field" figure. **Figure 2.P.** below will show the example of the listed coordinates by the "ans" variable.

The difference between the 2 figures above is that the first figure contains all the velocity vectors that are given by the loaded workspace variables. In this case, the loaded workspace variables contain 18 different data, each from a different streak particle. Thus, the first figure will plot 18 velocity vectors. However, the second figure (the filtered one) has a criterion that will make observers easier to separate for which size group of streaks they want to observe. This experiment used a criterion of the streak area which has an area of above 50, until less than 200 pixels. The criterion can be changed from the code's line

18. With this, there are only 5 streaks left shown by the second figure. For more details of pinned locations for each vector, please reffer to the **appendix** on the last page.



	1	2
1	94.5000	109.5000
2	292.5000	246.5000
3	438.5000	921.5000
4	516.5000	506.5000
5	558.5000	545.5000
6	574.5000	407.5000
7	601.5000	420.5000
8	612.5000	381.5000
9	630.5000	484.5000
10	708.5000	458.5000
11	868.5000	552.5000
12	1.1585e+03	702.5000
13	1.2455e+03	646.5000
14	1.2575e+03	658.5000
15	1.2925e+03	717.5000
16	1.2965e+03	623.5000
17	1.3385e+03	650.5000
18	1.3395e+03	579.5000

Figure 2.P. The origin coordinates for each plotted vector are in the “Experimental Curve-Fitted Velocity Field” figure.

The vectorplot code uses only the initial U and V velocity vectors to plot the velocity vectors. The code is also able to resolve the velocity vectors with precision and a good sense of the velocity streak’s shape (its direction is similar to the streak’s direction), relying on the results given by the streakprocess algorithm. In the figure below, a comparison between the raw original image and the resolved velocity vector plot will be shown.

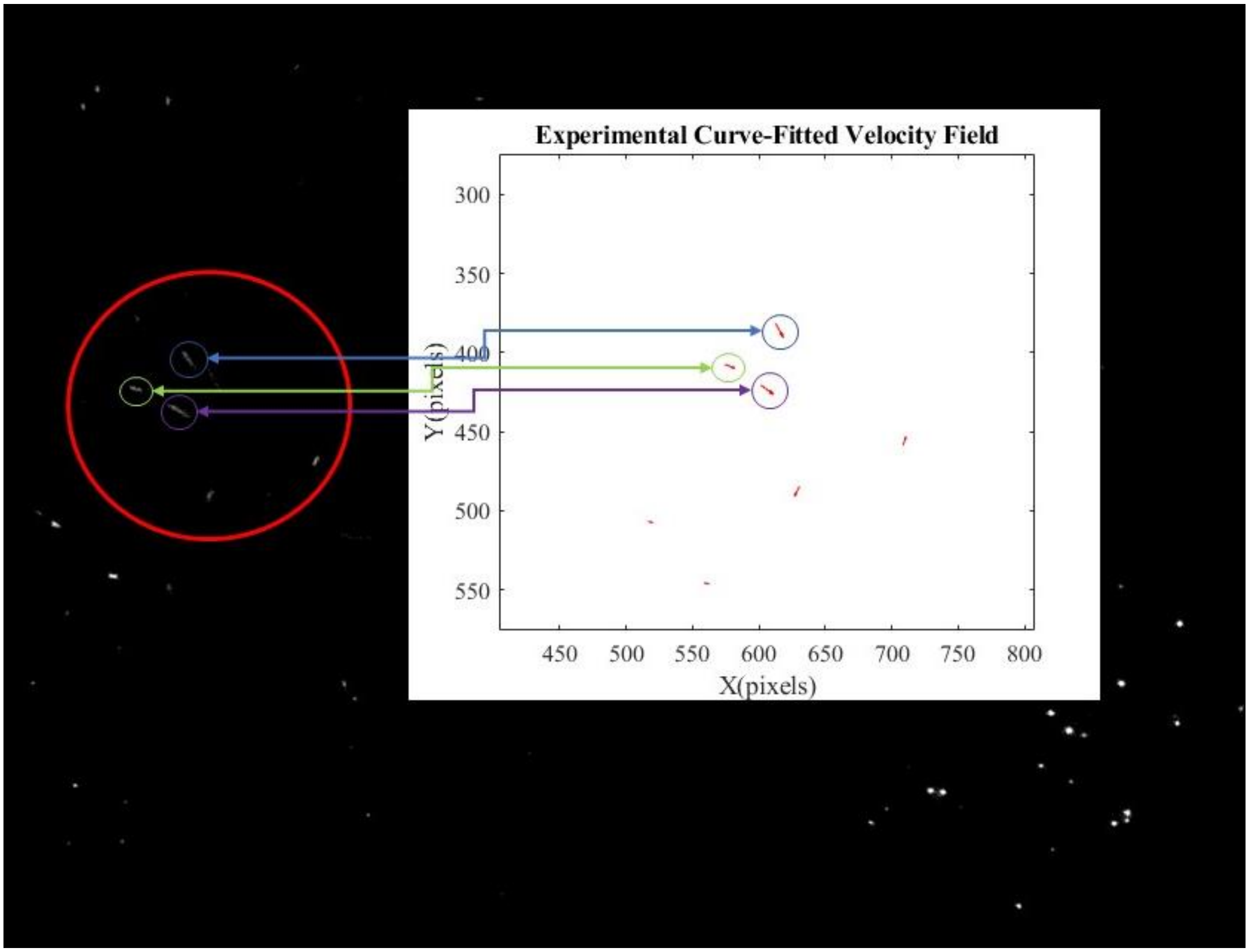


Figure 2.Q. Comparison between the original experimental image to the resolved velocity vector plot.

As seen in the figure above, the figure compares the three corresponding streaks to each other. The shape is well-aligned and is quite similar to one another. One problem that this algorithm hasn't solved perfectly is which point represents the start or the end of the streak, so the figure above can't justify yet if the streak's velocity direction is right, but the overall velocity vector's streak shape is assumed to be quite reliable. With the development of new techniques to indicate the start and endpoint of a streak, it is quite possible that this problem will be solved soon. One method that is planned now to solve this problem is to hold an experiment, where the illumination light keeps on decreasing/increasing its intensity value (making the light intensity as a function of time) so that we can see from the streak image what time does the streak's corresponding point is correlated to.

IV. Conclusions

From the data obtained from the streak-resolving algorithm, we may then plot the velocity vectors using the vector plotting code. Then we may observe the velocity vectors that are plotted by the code; and from the shown result diagram (**the last 4 appendix pages)** it is shown that the velocity vector has a sense of direction, similar to the original streak image's direction. This indicates that the resolved streak particle velocity components are quite accurate and reliable. The downside of the algorithm is that it doesn't have a streak start and endpoint detection, so the plan for the future is to make the illumination intensity during the image capturing section a function of time, so it is possible to know the correlation between the streak points and the corresponding time.

In the upcoming future, the use of the particle streak velocimetry method might be useful, because there might be a chance where the advancement of technology also creates inflation of hardware prices that might be a constraint for some researchers. By the development of this program which doesn't require advanced hardware, either to collect the experimental image data (doesn't need to use high-speed cameras) or to process the algorithm; it is desired that this will open more opportunity windows to do some research on this field.

V. References

I) 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*

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

III) Marcus Rosenstiel, Rolf-Rainer Grigat – *Segmentation and classification of streaks in a large-scale particle streak tracking system*

Appendix

1. Example for how to call the “streakprocess(image) code :

```
image = imread('Sample Crop Mr Mumtaz\2\2_main image.tif');
[I3_stats1,u_initial_pt_fit_list,u_middle_pt_fit_list,u_final_pt_fit_list,v_initial_pt_fit_list,v_middle_pt_fit_list,...
 v_final_pt_fit_list,Dia_fit,Amp_fit,Resnorm_List, Blob_filtered] = streakprocess(image);
```

2. Code of “streakprocess(image)” :

```
function [I3_stats1,u_initial_pt_fit_list,u_middle_pt_fit_list,u_final_pt_fit_list,v_initial_pt_fit_list,v_middle_pt_fit_list,v_final_pt_fit_list,Dia_fit,Amp_fit,Resnorm_List, Blob_filtered]=streakprocess(original)
close all;
warning off;
image_type = class(original);

% Show original image
figure;
imagesc(original);
% axis xy;
title('original');
% Top hat filter, for reducing background
se = strel('disk',5);
tophatFiltered = imtophat(original,se);
% Show top hat filtered image
figure
imagesc(tophatFiltered)
title('Tophat filtered')
% axis xy;
%%

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

% Apply the thresholding
I3=I2.*I2>thr;
% Show the thresholding result
figure
imagesc(I3);
```

```
title('thresholding');
% axis xy;
%%

% 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');
%NumberofStreaks = size(I3_stats1,1); %the additional variable I added for the iteration
%%
% At this point still use for loop to append image
% For each blob defined by tophatfiltering + thresholding, put the orginal pixel intensity values
back to the location
% Each blob image is saved in I3_stats1(i).Blob
% Bounding box coordinates can be used to restore the blob location in the
% original 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=30;
I3_Blob=zeros(size(I3));
I3_Region=zeros(size(I3));
for i=1:size(I3_stats1,1)
    Area=I3_stats1(i).Area;
    BBox=I3_stats1(i).BoundingBox;
    if Area>=thr_area
        I3_Blob(ceil(BBox(2)):ceil(BBox(2))+BBox(4)-1,ceil(BBox(1)):ceil(BBox(1))+BBox(3)-
1)=I3_stats1(i).Blob;
        % Use intensity = i/400 to show each connected region (blob) in
        % different color in I3_Region
        I3_(ceil(BBox(2)):ceil(BBox(2))+BBox(4)-1,ceil(BBox(1)):ceil(BBox(1))+BBox(3)-
1)=I3_stats1(i).Blob;
        I3_Region(ceil(BBox(2)):ceil(BBox(2))+BBox(4)-
1,ceil(BBox(1)):ceil(BBox(1))+BBox(3)-1)=I3_stats1(i).Image*(i/400);
        I3_stats1(i).I3_Region=I3_Region;

    end
end

figure;
imagesc(I3_Blob);
title('Blob images');
% axis xy;
figure;
imagesc(I3_Region);
title('Blob region');
% axis xy;

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

idx = [I3_stats1.Area]>=thr_area;
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;

Blob= Blob_filtered;
MaxBlob= max(Blob(:));
[r, c]=find(Blob==MaxBlob,1);
[rr, cc] = size(Blob);
A = zeros(193);
I_size=[193,193];
A((I_size(1)+1)/2-r:(I_size(1)+1)/2-r+rr-1, (I_size(1)+1)/2-c:(I_size(1)+1)/2-c+cc-1)=Blob;
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

BLOB=A;
ones(I_size(1),I_size(2));
[X,Y]=meshgrid(1:I_size(1),1:I_size(2));
xdata(:,:,1)=X;
xdata(:,:,2)=Y;
ydata=BLOB;

x001=[0.5, 94,   0.9,      94,     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,    94,   0.8,       94,     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,    102,   1.2,    102,     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];


[x1,resnorm1]= lsqcurvefit(@D2GaussFunction,x001,xdata,ydata,lb,ub)


px1_fit=[x1(6)   x1(7)   x1(8)   x1(9)   x1(10)   x1(11)   x1(12)    x1(13)];
py1_fit=[x1(14)   x1(15)   x1(16)   x1(17)   x1(18)   x1(19)   x1(20)    x1(21)];

px1_der_fit(n,1:7) = polyder(px1_fit);
py1_der_fit(n,1:7) = polyder(py1_fit);

u1_initial_pt_fit(n)=polyval(px1_der_fit(n,1:7),0);
v1_initial_pt_fit(n)=polyval(py1_der_fit(n,1:7),0);

u1_middle_pt_fit(n)=polyval(px1_der_fit(n,1:7),0.5);
v1_middle_pt_fit(n)=polyval(py1_der_fit(n,1:7),0.5);

u1_final_pt_fit(n)=polyval(px1_der_fit(n,1:7),1);
v1_final_pt_fit(n)=polyval(py1_der_fit(n,1:7),1);

Fittedanswers1 = x1;
Error1 = resnorm1;
```

```
x002=[0.5, 94,   0.9,    94,   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,     94,   0.8,    94,    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,    102,  1.2,  102,    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];

[x2,resnorm2]= lsqcurvefit(@D2GaussFunction,x002,xdata,ydata,lb,ub)

px2_fit=[x2(6)   x2(7)   x2(8)   x2(9)   x2(10)   x2(11)   x2(12)    x2(13)];
py2_fit=[x2(14)   x2(15)   x2(16)   x2(17)   x2(18)   x2(19)   x2(20)    x2(21)];

px2_der_fit(n,1:7) = polyder(px2_fit);
py2_der_fit(n,1:7) = polyder(py2_fit);

u2_initial_pt_fit(n)=polyval(px2_der_fit(n,1:7),0);
v2_initial_pt_fit(n)=polyval(py2_der_fit(n,1:7),0);

u2_middle_pt_fit(n)=polyval(px2_der_fit(n,1:7),0.5);
v2_middle_pt_fit(n)=polyval(py2_der_fit(n,1:7),0.5);

u2_final_pt_fit(n)=polyval(px2_der_fit(n,1:7),1);
v2_final_pt_fit(n)=polyval(py2_der_fit(n,1:7),1);

Fittedanswers2 = x2;
Error2 = resnorm2;


x003=[0.5,  89,   0.9,   89  , 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,     89,    0.8,     89  ,  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,     115,   1.2,   115  , 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];


[x3,resnorm3]= lsqcurvefit(@D2GaussFunction,x003,xdata,ydata,lb,ub)

px3_fit=[x3(6)   x3(7)   x3(8)   x3(9)   x3(10)   x3(11)   x3(12)    x3(13)];
py3_fit=[x3(14)   x3(15)   x3(16)   x3(17)   x3(18)   x3(19)   x3(20)    x3(21)];

px3_der_fit(n,1:7) = polyder(px3_fit);
py3_der_fit(n,1:7) = polyder(py3_fit);
```

```
u3_initial_pt_fit(n)=polyval(px3_der_fit(n,1:7),0);
v3_initial_pt_fit(n)=polyval(py3_der_fit(n,1:7),0);

u3_middle_pt_fit(n)=polyval(px3_der_fit(n,1:7),0.5);
v3_middle_pt_fit(n)=polyval(py3_der_fit(n,1:7),0.5);

u3_final_pt_fit(n)=polyval(px3_der_fit(n,1:7),1);
v3_final_pt_fit(n)=polyval(py3_der_fit(n,1:7),1);

Fittedanswers3 = x3;
Error3 = resnorm3;


x004=[0.5,  89,   0.9,  89,    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,      89,   0.8,   89,     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,     115,   1.2,  115  , 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];

[x4,resnorm4]= lsqcurvefit(@D2GaussFunction,x004,xdata,ydata,lb,ub)

px4_fit=[x4(6)   x4(7)   x4(8)   x4(9)   x4(10)   x4(11)   x4(12)    x4(13)];
py4_fit=[x4(14)   x4(15)   x4(16)   x4(17)   x4(18)   x4(19)   x4(20)    x4(21)];

px4_der_fit(n,1:7) = polyder(px4_fit);
py4_der_fit(n,1:7) = polyder(py4_fit);

u4_initial_pt_fit(n)=polyval(px4_der_fit(n,1:7),0);
v4_initial_pt_fit(n)=polyval(py4_der_fit(n,1:7),0);

u4_middle_pt_fit(n)=polyval(px4_der_fit(n,1:7),0.5);
v4_middle_pt_fit(n)=polyval(py4_der_fit(n,1:7),0.5);

u4_final_pt_fit(n)=polyval(px4_der_fit(n,1:7),1);
v4_final_pt_fit(n)=polyval(py4_der_fit(n,1:7),1);

Fittedanswers4 = x4;
Error4 = resnorm4;


x005=[0.5, 89, 0.9,  89  , 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,   89,   0.8,    89  ,  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,  104 ,   1.2,  104  , 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];


[x5,resnorm5]= lsqcurvefit(@D2GaussFunction,x005,xdata,ydata,lb,ub)

px5_fit=[x5(6)   x5(7)   x5(8)   x5(9)   x5(10)   x5(11)   x5(12)    x5(13)];
py5_fit=[x5(14)   x5(15)   x5(16)   x5(17)   x5(18)   x5(19)   x5(20)    x5(21)];

px5_der_fit(n,1:7) = polyder(px5_fit);
py5_der_fit(n,1:7) = polyder(py5_fit);

u5_initial_pt_fit(n)=polyval(px5_der_fit(n,1:7),0);
v5_initial_pt_fit(n)=polyval(py5_der_fit(n,1:7),0);

u5_middle_pt_fit(n)=polyval(px5_der_fit(n,1:7),0.5);
v5_middle_pt_fit(n)=polyval(py5_der_fit(n,1:7),0.5);

u5_final_pt_fit(n)=polyval(px5_der_fit(n,1:7),1);
v5_final_pt_fit(n)=polyval(py5_der_fit(n,1:7),1);

Fittedanswers5 = x5;
Error5= resnorm5;

x006=[0.5,  89,   0.9,   89  , 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,     89,    0.8,     89  ,  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,     115,   1.2,   115  , 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];

[x6,resnorm6]= lsqcurvefit(@D2GaussFunction,x006,xdata,ydata,lb,ub)

px6_fit=[x6(6)   x6(7)   x6(8)   x6(9)   x6(10)   x6(11)   x6(12)    x6(13)];
py6_fit=[x6(14)   x6(16)   x6(16)   x6(17)   x6(18)   x6(19)   x6(20)    x6(21)];

px6_der_fit(n,1:7) = polyder(px6_fit);
py6_der_fit(n,1:7) = polyder(py6_fit);

u6_initial_pt_fit(n)=polyval(px6_der_fit(n,1:7),0);
v6_initial_pt_fit(n)=polyval(py6_der_fit(n,1:7),0);

u6_middle_pt_fit(n)=polyval(px6_der_fit(n,1:7),0.5);
```

```
v6_middle_pt_fit(n)=polyval(py6_der_fit(n,1:7),0.5);

u6_final_pt_fit(n)=polyval(px6_der_fit(n,1:7),1);
v6_final_pt_fit(n)=polyval(py6_der_fit(n,1:7),1);

Fittedanswers6 = x6;
Error6 = resnorm6;


x007=[0.5,  89,   0.9,    89,    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,     89,    0.8,     89,   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,     115,   1.2,   115  , 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];


[x7,resnorm7]= lsqcurvefit(@D2GaussFunction,x007,xdata,ydata,lb,ub)

px7_fit=[x7(6)   x7(7)   x7(8)   x7(9)   x7(10)   x7(11)   x7(12)    x7(13)];
py7_fit=[x7(14)   x7(16)   x7(17)   x7(17)   x7(18)   x7(19)   x7(20)    x7(21)];

px7_der_fit(n,1:7) = polyder(px7_fit);
py7_der_fit(n,1:7) = polyder(py7_fit);

u7_initial_pt_fit(n)=polyval(px7_der_fit(n,1:7),0);
v7_initial_pt_fit(n)=polyval(py7_der_fit(n,1:7),0);

u7_middle_pt_fit(n)=polyval(px7_der_fit(n,1:7),0.5);
v7_middle_pt_fit(n)=polyval(py7_der_fit(n,1:7),0.5);

u7_final_pt_fit(n)=polyval(px7_der_fit(n,1:7),1);
v7_final_pt_fit(n)=polyval(py7_der_fit(n,1:7),1);


Fittedanswers7 = x7;
Error7 = resnorm7;


x008=[0.5, 94,   0.9,    94,    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,     94,   0.8,    94,   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];
```

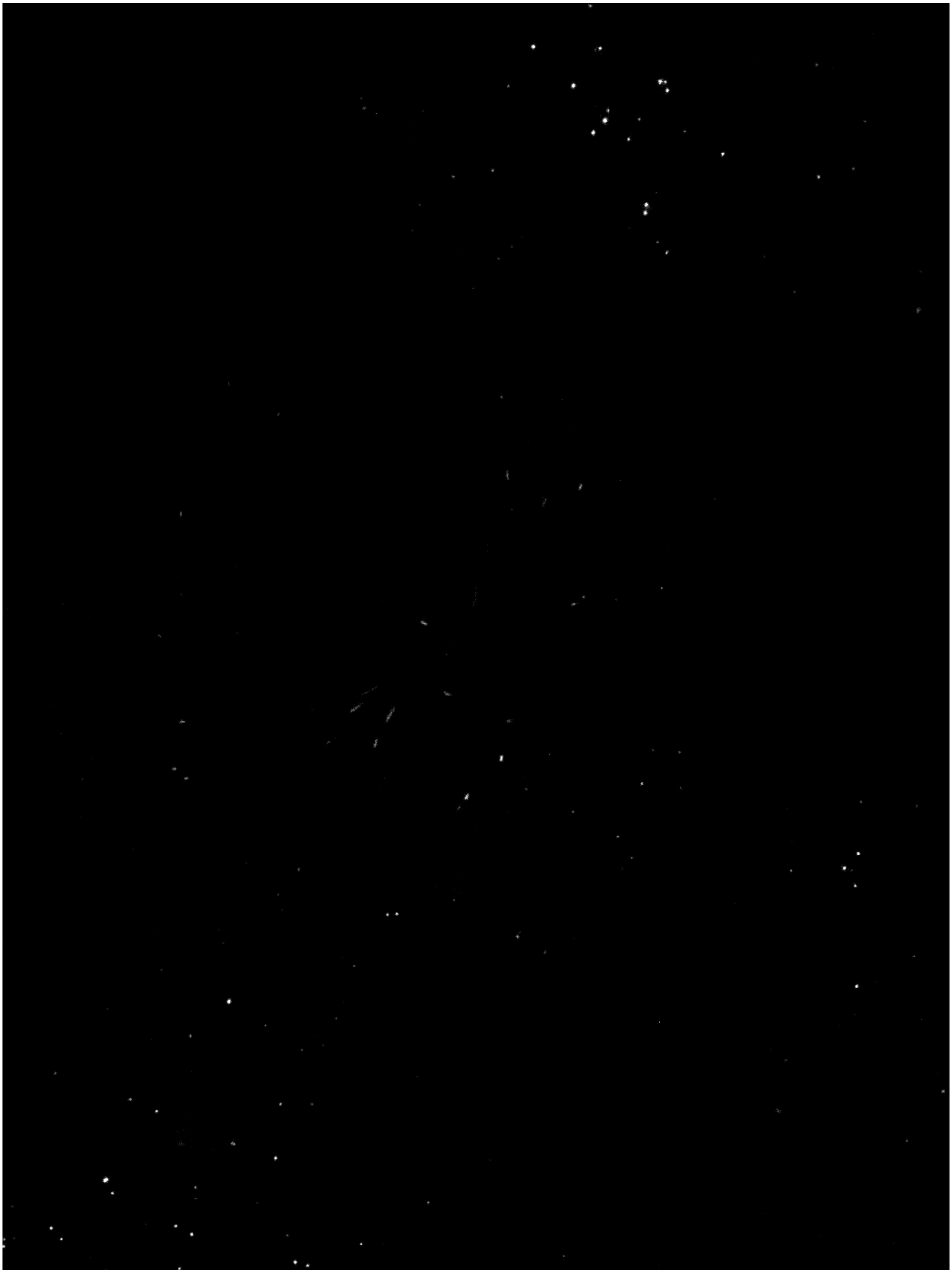
```
ub=[1.1,    102,  1.2,  102,   1,    3e-09,    1.1e-08,    3.2e-09,   2.05e-06,      5.5e-05,    0.0095,
1,     0 ,  2e-10,    7e-10,  -4e-08,    -9.99e-07,    0.0003,    0.0095,      1,    0];

[x8,resnorm8]= lsqcurvefit(@D2GaussFunction,x008,xdata,ydata,lb,ub)

px8_fit=[x8(6)      x8(7)    x8(8)     x8(9)     x8(10)   x8(11)   x8(12)    x8(13)];
py8_fit=[x8(14)   x8(16)   x8(17)   x8(17)   x8(18)   x8(19)   x8(20)    x8(21)];

px8_der_fit(n,1:7) = polyder(px8_fit);
py8_der_fit(n,1:7) = polyder(py8_fit);

u8_initial_pt_fit(n)=polyval(px8_der_fit(n,1:7),0);
v8_initial_pt_fit(n)=polyval(py8_der_fit(n,1:7),0);

u8_middle_pt_fit(n)=polyval(px8_der_fit(n,1:7),0.5);
v8_middle_pt_fit(n)=polyval(py8_der_fit(n,1:7),0.5);

u8_final_pt_fit(n)=polyval(px8_der_fit(n,1:7),1);
v8_final_pt_fit(n)=polyval(py8_der_fit(n,1:7),1);

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(:,6:13);
px_der_fit(n,:)=polyder(px_fit(n,1:8));
py_fit(n,1:8)=Fittedanswers(:,14:21);
py_der_fit(n,:)=polyder(py_fit(n,1:8));

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;
```

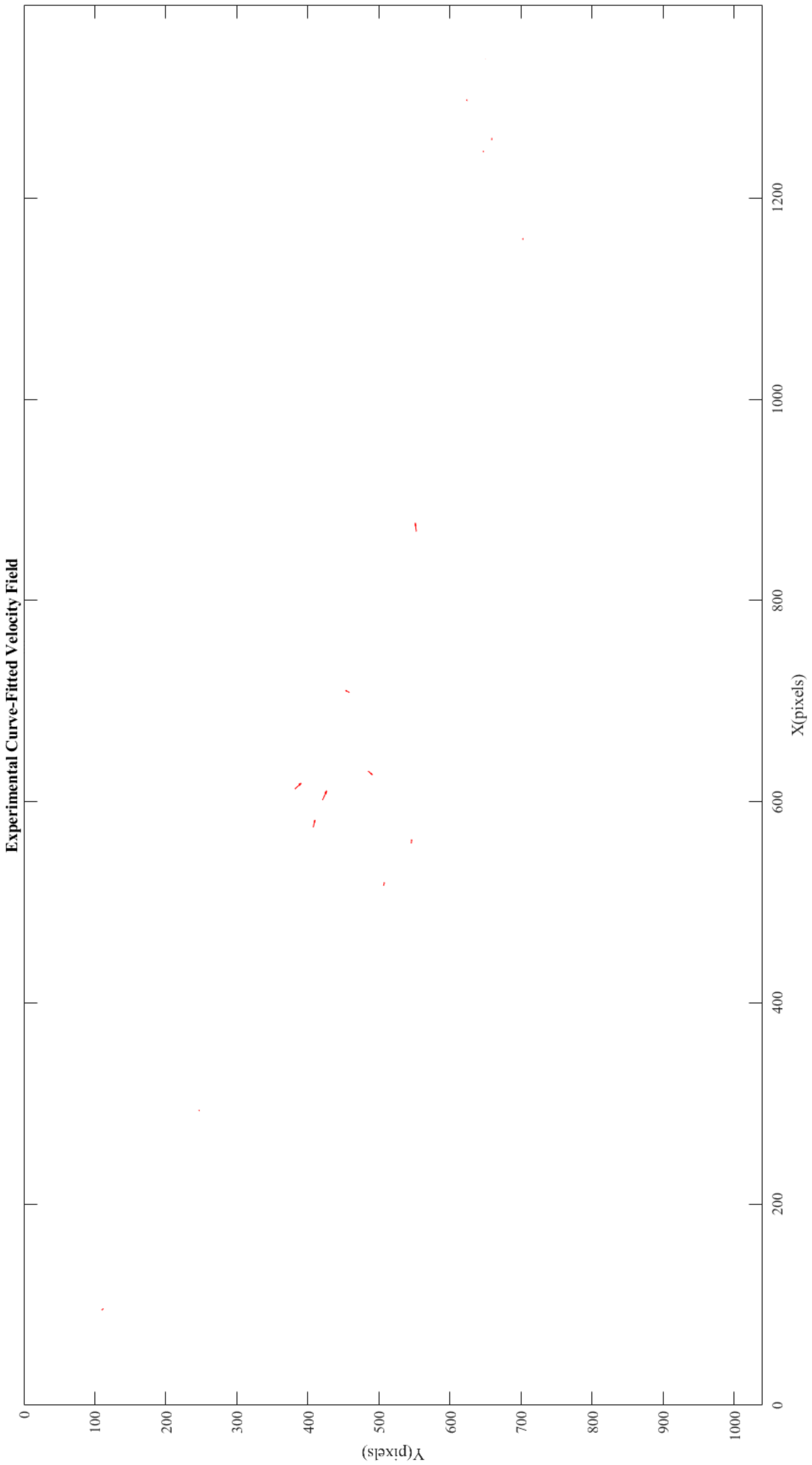
```
Amp_fit(n)=Fittedanswers(:,1);
Resnorm_List(n)=Fitteddata(n, 22);
fprintf('Just finished iteration #%d\n', counter);
counter=counter+1;
end
%% RESULTS

u_initial_pt_fit_list=reshape(u_initial_pt_fit,[],1);
u_middle_pt_fit_list=reshape(u_middle_pt_fit,[],1);
u_final_pt_fit_list=reshape(u_final_pt_fit,[],1);

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

Dia_fit=reshape(Dia_fit,[],1);
Amp_fit=reshape(Amp_fit,[],1);
Resnorm_List=reshape(Resnorm_List,[],1);


function F = D2GaussFunction(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);
```

3. Code of “vectorplot” :

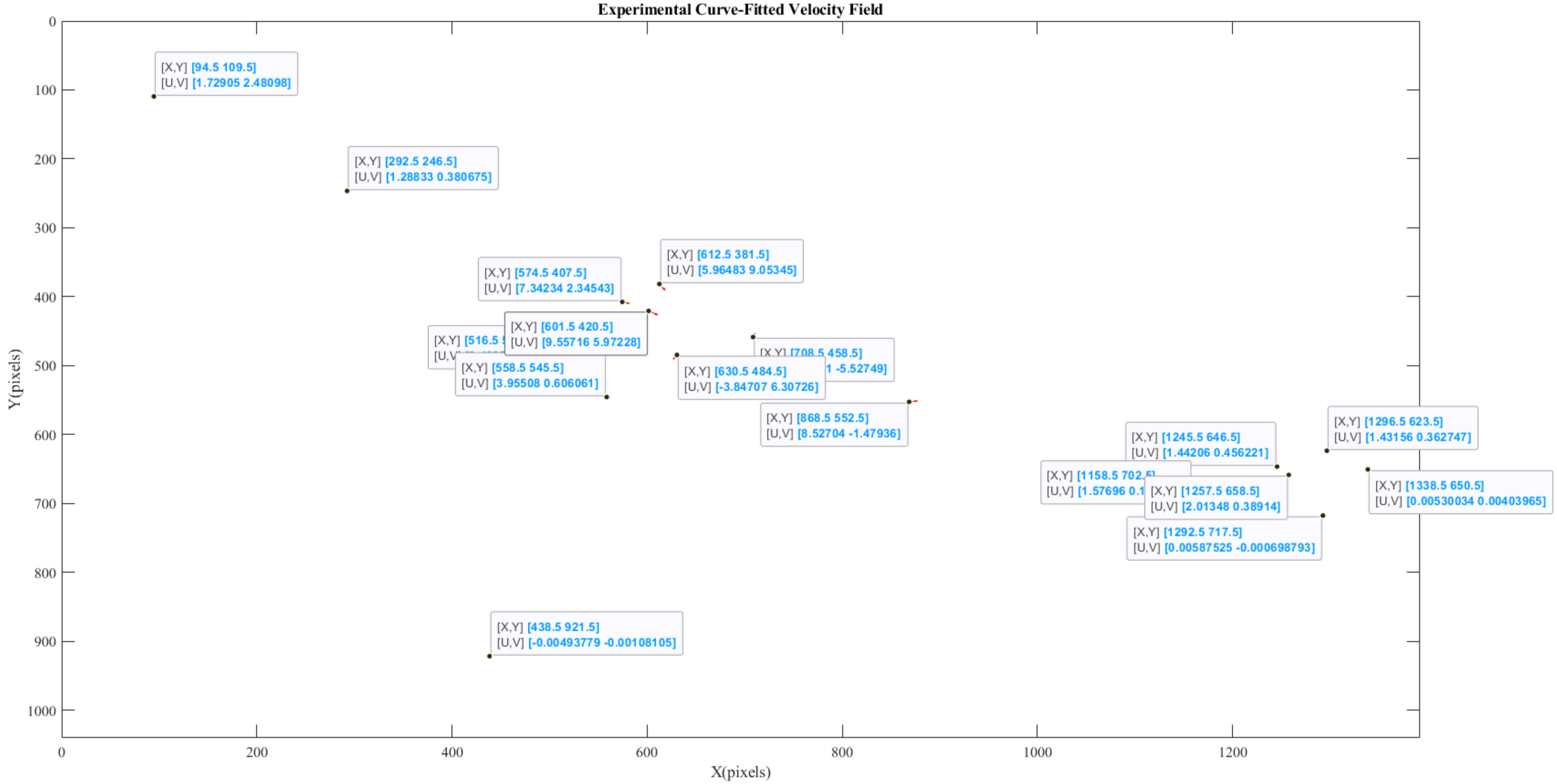
```
function
[Fitted_Streak_Positions,Fitted_Streak_Positions2,u_initial_pt_fit_list,v_initial_pt_fit_list,I3_sta
ts1,final_u_filtered,final_v_filtered,Final_Resnorm]=vectorplot;

close all;
load ('u_initial_pt_fit_list.mat');
u_initial_pt_fit_list;
load ('v_initial_pt_fit_list.mat');
v_initial_pt_fit_list;
load ('I3_stats1.mat');
thr_area=30;
idx = [I3_stats1.Area]>=thr_area;
idx=reshape(idx,[],1);
a=(1:length(idx))';
b=a.*idx;
Blob_filtered_idx=b(all(b,2),:);

thr_area2=[I3_stats1.Area]';
thr_area22=thr_area2((50 < thr_area2) & (thr_area2<200));
idx2=ismember(thr_area2,thr_area22);
idx2=reshape(idx2,[],1);
a=(1:length(idx2))';
b=a.*idx2;

Blob_filtered_idx2=b(all( b,2),:);

finalidx=ismember(Blob_filtered_idx,Blob_filtered_idx2);
final_u=finalidx.*u_initial_pt_fit_list;
final_v=finalidx.*v_initial_pt_fit_list;
out1=final_u(all(final_u,2),:);
out2=final_v(all(final_v,2),:);
final_u_filtered=out1;
final_v_filtered=out2;
load ('Resnorm_List.mat');
Resnorm=finalidx.*Resnorm_List;
out3=Resnorm(all( Resnorm,2),:);
Final_Resnorm=out3;

for i=1:length(Blob_filtered_idx)


   Blob_filtered_idxx=Blob_filtered_idx(i);
   BoundingBox_filtered=I3_stats1(Blob_filtered_idxx).BoundingBox;
   Streak_Locations_x=BoundingBox_filtered;
```

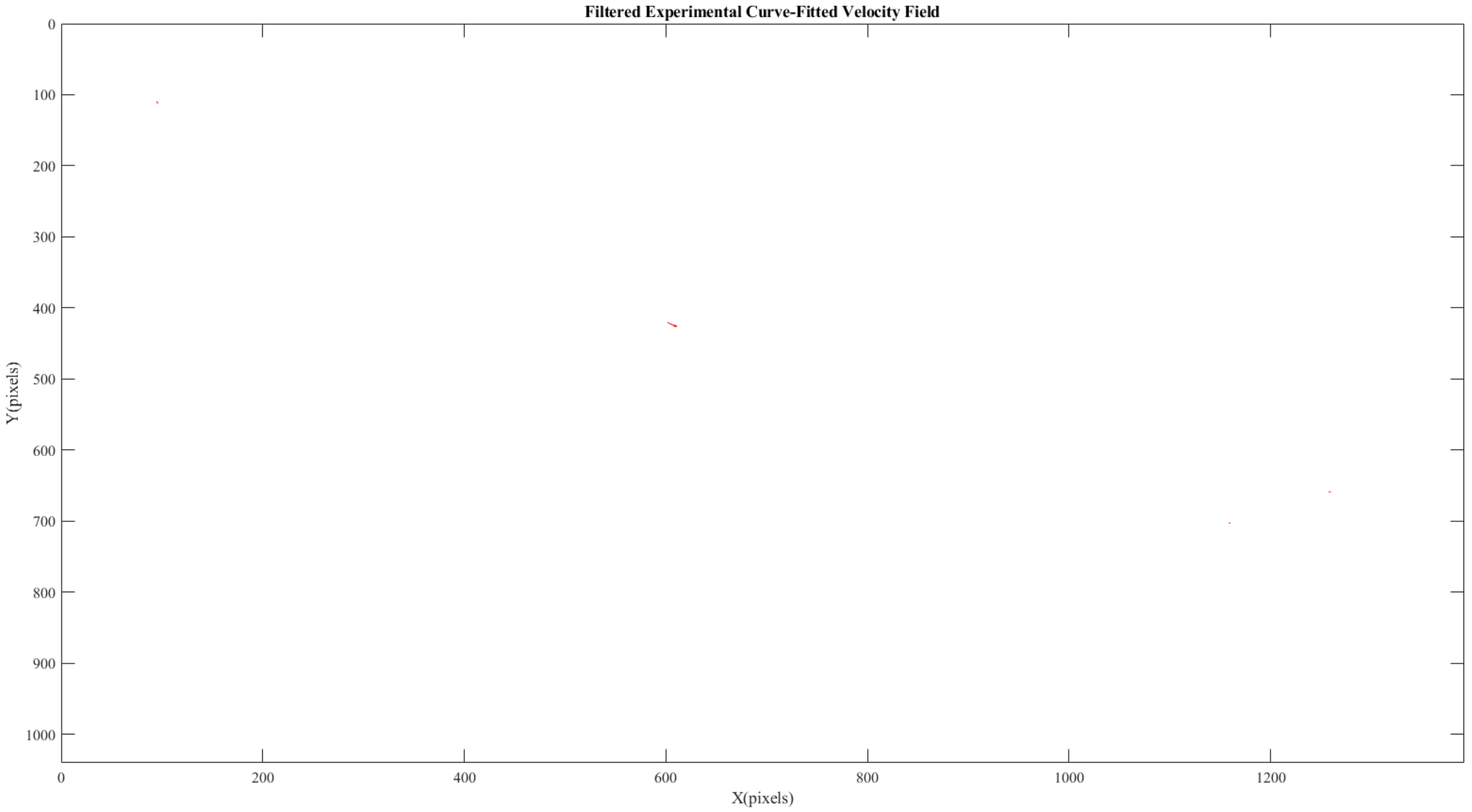
```
    Streak_Locations_y=BoundingBox_filtered;
    x(i,:)=Streak_Locations_x(1,1);
    y(i,:)=Streak_Locations_y(1,2);

end
x= reshape(x,[],1);
y= reshape(y,[],1);
Fitted_Streak_Positions=[x y];


for i=1:length(Blob_filtered_idx2)


    Blob_filtered_idx2x=Blob_filtered_idx2(i);
    BoundingBox_filtered=I3_stats1(Blob_filtered_idx2x).BoundingBox;
    Streak_Locations_xx=BoundingBox_filtered;
    Streak_Locations_yy=BoundingBox_filtered;
    xx(i,:)=Streak_Locations_xx(1,1);
    yy(i,:)=Streak_Locations_yy(1,2);

end
xx= reshape(xx,[],1);
yy= reshape(yy,[],1);
Fitted_Streak_Positions2=[xx yy];

figure(1);quiver(x,y, u_initial_pt_fit_list,(v_initial_pt_fit_list),0,'r')
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;

 figure(2);quiver(xx,yy, final_u_filtered,abs(final_v_filtered),0,'r')
xlim([0 1392]);
ylim([0 1040]);
xlabel('X(pixels)');
ylabel('Y(pixels)');
set ( gca, 'ydir', 'reverse' )
title('Filtered Experimental Curve-Fitted Velocity Field');
set(gca, 'FontName', 'Times')
ax = gca;
ax.FontSize = 12;
```

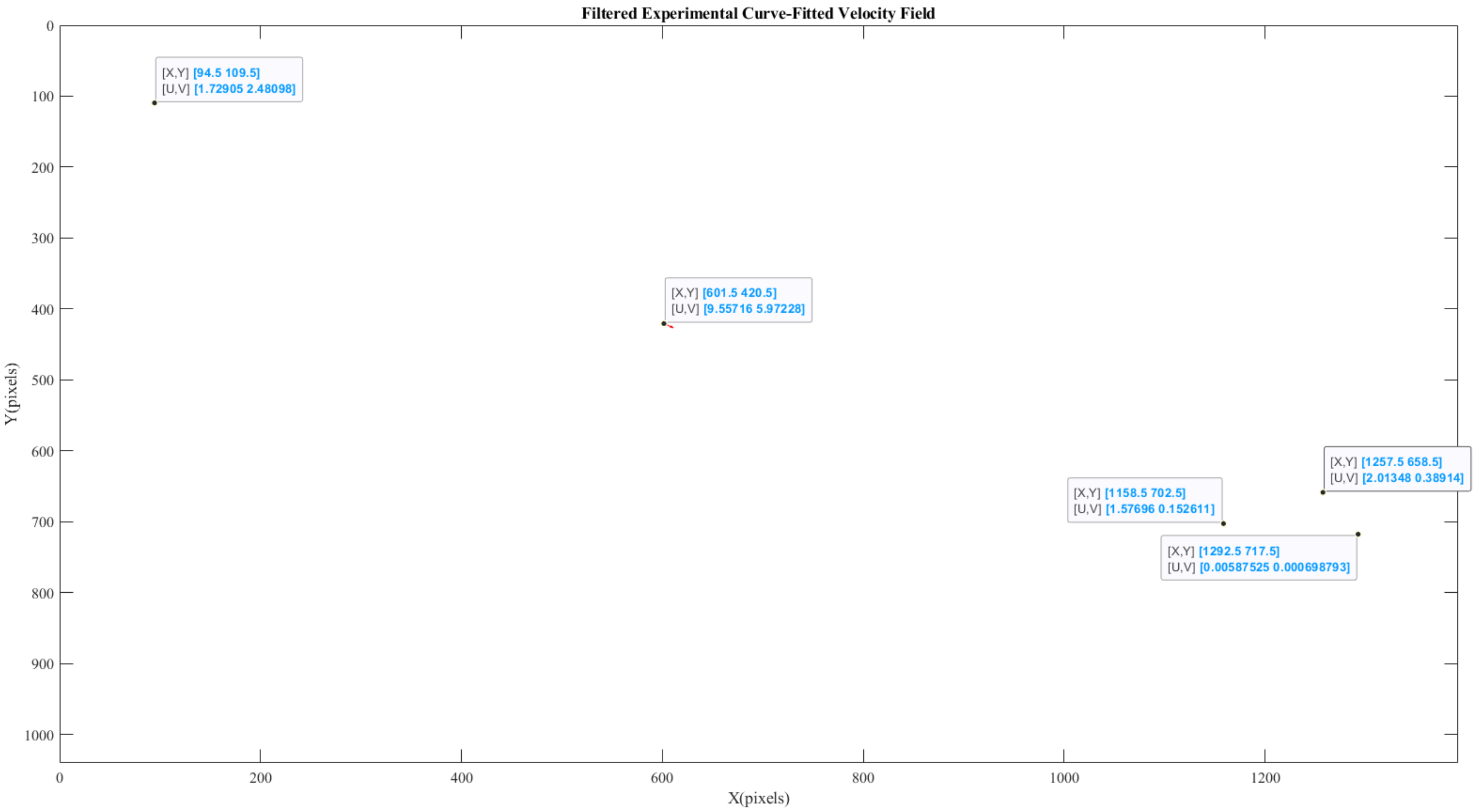



Experimental Curve-Fitted Velocity Field
X(pixels)
Y(pixels)
0
200
400
600
800
1000
1200
0
100
200
300
400
500
600
700
800
900
1000

Experimental Curve-Fitted Velocity Field




Filtered Experimental Curve-Fitted Velocity Field
Y(pixels)
0
100
200
300
400
500
600
700
800
900
1000
X(pixels)
0
200
400
600
800
1000
1200



Filtered Experimental Curve-Fitted Velocity Field