- Implement MATLINK coupling with MATLAB PIVlab https://www.mathworks.com/matlabcentral/fileexchange/27659-pivlab-particle-image-velocimetry-piv-tool-with-gui
- Frame_1 -> Frame_001
- Improved boundary/static artifact treatment
- GPU-based BM3D filtering instead of Perona-Malik
- GPU-based NMM instead of the CPU version

```
In[ ]:= ClearAll["Global`*"]
```

Important:

For seamless automatic processing, do this:
1) Disable (using "Alt + /" to (un-)comment) all import cells except image import
2) Use memorized crop bounds (instructions below)

For checking & re-doing existing input:
Option 1: place the experiment folder with run subfolders into the 'output' folder
Option 2: change the 'mainOutputFolder' in "Initialization" to the main results folder

---

ALWAYS EVALUATE 'Initialization' FIRST

Initialization

```
$HistoryLength = 0;
nbd := NotebookDirectory[]

(*packageDirectory:=
 "D:\\Science\\NMI (VTPMML)\\MHD Project\\Image Processing\\Mathematica
    Libraries & Packages\\Quantile Regression\\"

  Import[packageDirectory<>"QuantileRegression.m"];
<<MaTeX`*)


SetOptions[
  ListPlot,
  {Frame → True, FrameStyle → {Black, Bold}, TicksStyle → {Black, Bold},
   LabelStyle → {Black, Bold, fontSize}, ImageSize → Large}
 ];
```

```
SetOptions[
  ListLinePlot,

  {Frame → True, FrameStyle → {Black, Bold},
   TicksStyle → {Black, Bold}, LabelStyle → {Black, Bold, fontSize},
   PlotStyle → Automatic, Filling → None, ImageSize → Large}
 ];


launchNukes[useKernels_] := If[
      Length@Kernels[] < #,
      # - Length@Kernels[] // LaunchKernels,
      Nothing
     ] &@useKernels;


mainOutputFolder = nbd <> "Output";
CreateDirectory@mainOutputFolder; // Quiet


progress[x_, y_] := TableForm@
  {
   "Processing...",
   ProgressIndicator@Dynamic[x / y // N],
   ToString@Round@x <> "/" <> ToString@y,
   ToString@PercentForm[x / y // N, DefaultPrintPrecision → 1]
  }


memoryCheck[] :=
 "Memory in use: " <> ToString[MemoryInUse[] / 1024^3 // N] <> " Gb" // Print

memoryCleanupStart[] := Module[{},
  "Memory in use: " <> ToString[MemoryInUse[] / 1024^3 // N] <> " Gb" // Print;
  Unprotect@Out;
  CloseKernels[];
  ClearSystemCache[];
 ]

memoryCleanupFinalize[] := Module[{},
  Clear@Out;
  Protect@Out;
  LaunchKernels@useKernels;
  "Memory in use after cleanup: " <>
    ToString[MemoryInUse[] / 1024^3 // N] <> " Gb" // Print
 ]


useKernels = 2 * $ProcessorCount - 2;
launchNukes@useKernels;
SetSharedVariable@monitorCounter;
```

>> Import Images <<

Set the raw image folder:

imageFolder -- main folder with all experiments
subFolder -- experiment folder (all runs), i.e. “5p_ 4ul-min”

avoid extra pathname separators at the end of path strings

In[ ]:=
```
(*pixelSizeMicroMeters=0.064;
frameDurationMiliSeconds=50;
frameRate=1000/frameDurationMiliSeconds;*)

imageFolder := "D:\\[BIG DATA]\\ACTIVE - Data
   From Experiments\\MMML Data\\OOC Particle Flow\\[RAKSTAM]"

subFolder := "5p_4ul-min";
fileFolder = SetDirectory[imageFolder <> $PathnameSeparator <> subFolder];

First@StringSplit[#, "."] &@# & /@ SortBy[
     FileNames["*.tif"],
     ToExpression@FileBaseName@# &
    ] // DeleteDuplicates // Sort
```

runID -- select the imaging run (focus plane) for processing

Example:
runID=4;

In[]:=
```
runID = 6;

runIdentifier = "run" <> ToString@runID <> ".";

files = Select[#, StringContainsQ@runIdentifier] &@SortBy[
    FileNames["*.tif"],
    ToExpression@FileBaseName@# &
   ];

(*files=Sort@FileNames["*.tiff"]*)

files // Length
files[[1 ;; 25]]

output = mainOutputFolder <> $PathnameSeparator <> subFolder <>
   $PathnameSeparator <> StringDelete[runIdentifier, "."] <> $PathnameSeparator;
CreateDirectory@output; // Quiet

dataCSVfolder = output <> "CSV";
CreateDirectory@dataCSVfolder; // Quiet

dataCompressedfolder = output <> "Compressed Data";
CreateDirectory@dataCompressedfolder; // Quiet
```

In[]:=
```
monitorCounter = 0;

Monitor[
 images = ParallelMap[
    (monitorCounter++; Import[
       fileFolder <> $PathnameSeparator <> #,
       IncludeMetaInformation → None
      ]) &, files
   ];, progress[monitorCounter, Length@files]
]

images = Last /@ images;
#@First@images & /@ {ImageDimensions, ImageType} // TableForm
```

Function Definitions

ALWAYS EVALUATE THESE CELLS

Image Filtering & Manipulation

```
backgroundBias = 10^-6;
placeholderBias = 10^-6;
```

```
viewImage[image_] := Image[image, ImageSize → Medium]
rotateImage[image_] := ImageRotate[#, rotationSwitch * π / 2] &@image

meanCorrection[image_, meanImage_] := image - meanImage // ImageAdjust

meanCorrectionFlat[image_, meanImage_] :=
 ImageMultiply[image, meanImage^(-1)] // ImageAdjust


applySCTMM[image_] := ImageAdjust@ImageMultiply[
      #, # - ColorNegate@ColorToneMapping[
         #, luminanceCompression, Method → "Luminance"
        ]] &@image

applyNMM[image_] := 2 * image - nonlocalCorrectionWeight * NonlocalMeansFilter[
      image, nonlocalCorrectionRadius, nonlocalNoiseFactor
     ] // ImageAdjust


preprocessImage[image_] := BrightnessEqualize[# + placeholderBias] &@
  ImageAdjust@ColorToneMapping@ImageAdjust@image

preprocessImageInverted[image_] := ColorNegate@preprocessImage@ColorNegate@image
applyCTMinverted[image_] := ColorNegate@ColorToneMapping@ColorNegate@image

globalFiltering[image_] := Nest[
            applyCTMinverted, #, preprocessPassesSecondary
           ] &@PeronaMalikFilter[
           #, peronaMalikIterations, peronaMalikConductivity
          ] &@applyNMM@Nest[ColorToneMapping, #, useCTMprimary] &@Nest[
       preprocessImageInverted, #, preprocessPassesPrimary
      ] &@ImageMultiply[ColorNegate@channelWallMask, #] &@image

globalSegmentation[image_] := SelectComponents[
         #, sizeMinimum < #Count < sizeMaximum &
        ] &@DeleteBorderComponents@Binarize[#, Method → "Entropy"] &@
    ImageMultiply[ColorNegate@channelWallMask, #] &@image


getImageDisplacementsMap[imagePair_] :=
 Image@Rescale@First@ImageDisplacements@imagePair


getChannelWallMask[image_] :=
 ImageSubtract[#, DeleteBorderComponents@#] &@Dilation[#, DiskMatrix@dilationScale] &@
         Closing[#, DiskMatrix@closingScale] &@
       Binarize@GaussianFilter[#, gaussianScale] &@LocalAdaptiveBinarize[
      applySCTMM@ColorNegate@#,
      adaptiveBinarizeScaleFactor * Median@ImageDimensions@# // N // Round
     ] &@ImageRotate[image, -rotationAngle]
```

Data Processing

```
exportImageSequence[images_, folder_, format_, compression_] := Module[
  {
   imageSequence = images, exportFormat = format,
   exportFolder = folder, compressionLevel = compression, k, monitorCounter
  },

  monitorCounter = 0;

   Monitor[
    For[k = 1, k ≤ Length@imageSequence, k++,
     Export[
      exportFolder <> $PathnameSeparator <>
       "Frame_" <> ToString@k <> exportFormat, imageSequence[[k]],
      CompressionLevel → compressionLevel
     ]; monitorCounter++;
    ], progress[monitorCounter, Length@imageSequence]
   ]; // AbsoluteTiming
 ]



exportImageSequenceWithSize[images_,
  folder_, format_, compression_, resolution_] := Module[
  {
   imageSequence = images, exportFormat = format, exportFolder = folder,
   compressionLevel = compression, imageResolution = resolution, k, monitorCounter
  },

  monitorCounter = 0;

   Monitor[
    For[k = 1, k ≤ Length@imageSequence, k++,
     Export[
      exportFolder <> $PathnameSeparator <>
       "Frame_" <> ToString@k <> exportFormat, imageSequence[[k]],
      CompressionLevel → compressionLevel, ImageResolution → imageResolution
     ]; monitorCounter++;
    ], progress[monitorCounter, Length@imageSequence]
   ]; // AbsoluteTiming
 ]
```

Plots

In[◦]:=
```
getMaskOverlays[image_, mask_, color_, thickness_] := ImageAdd[
  ImageAdjust@image,
  ColorReplace[#, White → color] &@Binarize@GaussianFilter[#, thickness] &@
   EdgeDetect@mask
 ]

getLatexStyling[latexInput_] := MaTeX[#, FontSize → fontSize + 2] &@(
     "\\boldsymbol{" <> ToString@# <> "}"
    ) &@latexInput
```

[[Image Preview]]

Defaults :

rotationSwitch=0;
rotationAngle=π/2;

In[◦]:=
```
rotationSwitch = 1;
rotationAngle = π / 2;
```

In[◦]:=
```
previewImages = images;

Manipulate[
 viewImage@rotateImage@(*ColorToneMapping@*)ImageAdjust@previewImages[[k]],
 {k, 1, Length@previewImages, 1}
]
```

Crop Images

Crop Images

Set Crop Boundaries

Defaults :

selectedImages=1;;All;

In[ ]:=
```
selectedImages = 1 ;; All;
```

```
sequenceImages = images[[selectedImages]];

Manipulate[

 testImage = ImageAdjust@sequenceImages[[imageID]];
{width, height} = ImageDimensions@testImage;

 Manipulate[
  Grid[
   {{
     Show[
      testImage,
      Graphics[
        {EdgeForm[{Yellow, Thickness@0.0035}],
         Opacity@0.10, Yellow, Rectangle@@points}
      ], ImageSize → 525
     ]},
    {
     Show[
      croppedImage = ImageTake[
         testImage,
         rowRange = Reverse[height - points[[All, 2]]],
         columnRange = points[[All, 1]]
        ], ImageSize → UpTo@450
     ]}}],
  {
   {points, {{0, 0}, {width, height} / 2}}, Locator
  }],
 {imageID, 1, Length@sequenceImages, 1}
]
```

Crop Images

Optional:

For experiments with identical FOV for all runs,
evaluate '{rowRange,columnRange}' after setting crop boundaries;
copy the output to the 'memorizedRowColumnRanges' definition
Comment (disable, with "Alt + /", enabling again works the same) the crop setter module
Leave the 'selectedImages' and 'sequenceImages' definitions active

```
(*{rowRange,columnRange}*)
(*memorizedRowColumnRanges={{0,1024},{350,847}};*)
```

```
(*{rowRange,columnRange}=memorizedRowColumnRanges*)
{rowRange, columnRange} // TableForm

Manipulate[
 rotateImage@ImageAdjust@Image[#, ImageSize → Medium] &@ImageTake[
   sequenceImages[[k]],
   rowRange, columnRange
  ],
 {k, 1, Length@sequenceImages, 1}
]
```

Defaults :

imageIndices=1;;All;

In[◦]:=
```
imageIndices = 1 ;; All;
```

In[◦]:=
```
croppedImages = ImageTake[#, rowRange, columnRange] & /@
     sequenceImages[[imageIndices]] // Parallelize; // AbsoluteTiming

First@croppedImages // ImageDimensions // TableForm
```

<< Export Cropped Images >>

In[◦]:=
```
croppedImagesFolder = output <> "1 Cropped Images";
croppedImagesFolder // CreateDirectory; // Quiet

exportImageSequence[
   croppedImages, croppedImagesFolder, ".tiff", 0
  ]; // AbsoluteTiming
```

>> Import Cropped Images <<

```
croppedImagesFolder = output <> "1 Cropped Images";
launchNukes@useKernels;
SetDirectory@croppedImagesFolder;

files = SortBy[
   FileNames[],
   ToExpression@FileBaseName@# &
  ];

monitorCounter = 0;

Monitor[
 croppedImages = ParallelMap[
     (monitorCounter++; Import[
        croppedImagesFolder <> $PathnameSeparator <> #,
        IncludeMetaInformation → None
       ]) &, files
   ];, progress[monitorCounter, Length@files]
]
```

Image Projections & Correction

Image Stack Projections

Defaults :

rotationSwitch=1;
rotationAngle=π/2;
checkupIDs=1;;All;

In[ ]:=
```
rotationSwitch = 1;
rotationAngle = π / 2;
checkupIDs = 1 ;; All;
```

```
In[ ]:= imageStack = rotateImage@Image[ColorCombine@#, "Bit16"] &@croppedImages[[checkupIDs]]; //
         AbsoluteTiming
        ToString@N[ByteCount@imageStack / 1024^3] <> " Gb" // Print
        memoryCheck[]

        imageForms = Image[#, Byte] & /@ ImageAdjust /@ {
                rotateImage@First@#,
                rotateImage@Mean@#,
                ImageApply[Min, #] &@imageStack,
                ImageApply[StandardDeviation, #] &@imageStack
               } &@croppedImages[[checkupIDs]]; // AbsoluteTiming
```

Defaults :

rotationSwitch=1;
useCTMprojections=1;

```
In[ ]:= rotationSwitch = 1;
        useCTMprojections = 1;
```

```
In[ ]:= plotImageForms = GraphicsGrid[
          Rasterize[
                #, RasterSize → 0.5 * ImageDimensions@rotateImage@First@images
               ] & /@ Nest[ColorToneMapping, #, useCTMprojections] & /@ imageForms //
           Partition[#, 2] &,
          Dividers → All, ImageSize → Full, Dividers → All, Spacings → 5
         ]
```

```
In[ ]:= Export[
          output <> StringDelete[subFolder, $PathnameSeparator] <>
           "_OG_mean_min_stdev_projections.jpg",
          plotImageForms, CompressionLevel → 0
         ];

        memoryCleanupStart[]
        ClearAll /@ Unevaluated[{imageStack, checkupIDs}];
        memoryCleanupFinalize[]
```

Particle Motion Check

In[ ]:=
```
motionCheckSingleImage =
  #^(5/2) &@ColorToneMapping@HistogramTransform@First@croppedImages // rotateImage;

motionCheckImageStack = ColorToneMapping@#^(5/2) &@ColorToneMapping[
    Last@#/#[[3]] &@imageForms
   ];

plotMotionCheck = GraphicsColumn[
  {motionCheckSingleImage, motionCheckImageStack},
  ImageSize → Full, Dividers → All, Spacings → Scaled@0.025
 ]
```

In[ ]:=
```
Export[
  output <> StringDelete[subFolder, $PathnameSeparator] <> "_motion_check.jpg",
  plotMotionCheck, CompressionLevel → 0
 ];
```

Image Correction (Optional)

Defaults :

useCTMcorrection=0; << 0 or 1
meanCorrectionInvervals={1;;All};
useFlatCorrection=0;

In[ ]:=
```
useCTMcorrection = 1;
meanCorrectionInvervals = {1 ;; All};
useFlatCorrection = 1;
Length@meanCorrectionInvervals
```

```
In[]:= meanImages = Mean@croppedImages[[#]] & /@ meanCorrectionInvervals; // AbsoluteTiming
monitorCounter = 0;

Monitor[
  imagesCorrected = Flatten@Table[
      ParallelMap[
       (monitorCounter++;
         Nest[ColorToneMapping, #, useCTMprojections] &@ColorNegate@If[
              useFlatCorrection == 1, meanCorrectionFlat, meanCorrection
             ][
             ColorNegate@#, meanImages[[interval]]
            ]) &, croppedImages[[#]] &@meanCorrectionInvervals[[interval]]
       ], {interval, 1, Length@meanCorrectionInvervals}
     ];, progress[monitorCounter, Length@Flatten@croppedImages]
 ] // AbsoluteTiming

In[]:= Manipulate[
  viewImage@rotateImage@ImageAdjust@imagesCorrected[[k]],
  {k, 1, Length@imagesCorrected, 1}
 ]
```

Particle CNR Boost & Detection

Segment Channel Boundaries

Things to adjust if necessary:

‘luminanceCompression’ -- for initial boundary segmentation
‘dilationScale’ -- controls the boundary-conformal cutoff buffer for artifact mitigation
‘adaptiveBinarizeScaleFactor’ -- for smoother (straighter) or finer-detailed boundary

‘detectChannelWalls’ -- set to 0 if the entire FOV is relevant, otherwise set to 1 to detect channel walls and create buffers

Defaults :

detectChannelWalls=0; << 0 or 1

luminanceCompression=0.5;
adaptiveBinarizeScaleFactor=0.5;
gaussianScale=2;

closingScale=5;
dilationScale=10;

In[∘]:=
```
detectChannelWalls = 1;

luminanceCompression = 0.4;
adaptiveBinarizeScaleFactor = 0.75;
gaussianScale = 2;
closingScale = 5;
dilationScale = 20;
```

In[∘]:=
```
channelWallMask = If[
      detectChannelWalls == 1,
      channelWallMask = getChannelWallMask@#,
      channelWallMask = Binarize[#, 1] &@ImageRotate[#, -rotationAngle]
     ] &@imageForms[[3]];

plotChannelWalls = GraphicsColumn[
   getMaskOverlays[
      #, ImageRotate[#, rotationAngle] &@channelWallMask, Red, 2
     ] & /@ {
     imageForms[[3]],
     preprocessImageInverted@rotateImage@First@imagesCorrected
    }, ImageSize → Full, Dividers → All, Spacings → Scaled@0.025
 ]

channelWallMaskPhysical = Erosion[#, DiskMatrix@dilationScale] &@channelWallMask;

plotChannelWallsPhysical = getMaskOverlays[
      imageForms[[3]], rotateImage@#, Red, 2
     ] &@channelWallMaskPhysical;

viewImage@getMaskOverlays[#, rotateImage@channelWallMask, Yellow, 2] &@
 plotChannelWallsPhysical
```

<< Export Channel Wall Mask >>

In[∘]:=
```
MapThread[
   Export[output <> StringDelete[subFolder, $PathnameSeparator] <>
      "_channel_walls_mask_" <> #1, #2, CompressionLevel → 0] &, {
    {".jpg", "physical.jpg", "binary.png", "binary_physical.png"},
    {plotChannelWalls,
     plotChannelWallsPhysical, channelWallMask, channelWallMaskPhysical}
   }];
```

[[Test Filter Output]]

Things to adjust if necessary:

preprocessPassesSecondary -- usually anywhere between 10 an 20
Default: 15; for 8 flow rates, might want 20
For cases with channel walls outside the FOV, can set to 0

Defaults :

preprocessPassesPrimary=2;
useCTMprimary=0;

nonlocalCorrectionRadius=2;
nonlocalNoiseFactor=1;
nonlocalCorrectionWeight=1.75;

peronaMalikConductivity=0.1;
peronaMalikIterations=2;

preprocessPassesSecondary=15;
{sizeMinimum,sizeMaximum}={5,50};

In[ ]:=
```
preprocessPassesPrimary = 2;
useCTMprimary = 0;

nonlocalCorrectionRadius = 2;
nonlocalNoiseFactor = 1;
nonlocalCorrectionWeight = 1.75;

peronaMalikConductivity = 0.1;
peronaMalikIterations = 0;

preprocessPassesSecondary = 15;
{sizeMinimum, sizeMaximum} = {0, 20};
```

Options :

testImages=croppedImages;
testImages=imagesCorrected;

```
In[]:= testID = 1;
testImages = imagesCorrected;

stage0 =
  rotateImage@ImageMultiply[ColorNegate@channelWallMask, #] &@testImages[[testID]];

stage1 = Nest[ColorToneMapping, #, useCTMprimary] &@Nest[
      preprocessImageInverted, #, preprocessPassesPrimary
     ] &@stage0; // AbsoluteTiming

stage2 = applyNMM@stage1; // AbsoluteTiming

stage3 = PeronaMalikFilter[
     #, peronaMalikIterations, peronaMalikConductivity
    ] &@stage2; // AbsoluteTiming

stage4 = Nest[applyCTMinverted, #, preprocessPassesSecondary] &@stage3; // AbsoluteTiming

stage5 = SelectComponents[
        #, sizeMinimum < #Count < sizeMaximum &
       ] &@DeleteBorderComponents@Binarize[#, Method → "Entropy"] &@ImageMultiply[
      rotateImage@ColorNegate@channelWallMask, #] &@stage4; // AbsoluteTiming

Partition[#, 2] &@{
   stage0, stage1, stage2, stage3, stage4, stage5
  } // GraphicsGrid[#, ImageSize → Full, Dividers → All, Spacings → Scaled@0.025] &

HighlightImage[stage4, ImageMarker[stage5, "Circle"]]
```

Process Images

```
In[]:= monitorCounter = 0;

Monitor[
  imagesFiltered = ParallelMap[
     (monitorCounter++; globalFiltering@#) &, testImages
    ];, progress[monitorCounter, Length@testImages]
 ] // AbsoluteTiming

monitorCounter = 0;

Monitor[
  particleMasks = ParallelMap[
     (monitorCounter++; globalSegmentation@#) &, imagesFiltered
    ];, progress[monitorCounter, Length@imagesFiltered]
 ] // AbsoluteTiming
```

[[Preview Results]]

In[ ]:=
```
Manipulate[
 viewImage@rotateImage@imagesFiltered[[k]],
 {k, 1, Length@particleMasks, 1}
]

Manipulate[
 viewImage@rotateImage@particleMasks[[k]],
 {k, 1, Length@particleMasks, 1}
]
```

<< Export Filtered Images & Particle Masks >>

In[ ]:=
```
filteredImagesFolder = output <> "2 Filtered Images";
filteredImagesFolder // CreateDirectory; // Quiet

exportImageSequence[
   imagesFiltered, filteredImagesFolder, ".tiff", 0
  ]; // AbsoluteTiming

filteredImagesFolder = output <> "2 Filtered Images Compressed";
filteredImagesFolder // CreateDirectory; // Quiet
imagesFilteredCompressed = Compress /@ imagesFiltered // Parallelize; // AbsoluteTiming

exportImageSequence[
   imagesFilteredCompressed, filteredImagesFolder, ".txt", 1
  ]; // AbsoluteTiming

particleMasksFolder = output <> "3 Particle Masks";
particleMasksFolder // CreateDirectory; // Quiet

exportImageSequence[
   particleMasks, particleMasksFolder, ".tiff", 0
  ]; // AbsoluteTiming

particleMasksFolder = output <> "3 Particle Masks Compressed";
particleMasksFolder // CreateDirectory; // Quiet
particleMasksCompressed = Compress /@ particleMasks // Parallelize; // AbsoluteTiming

exportImageSequence[
   particleMasksCompressed, particleMasksFolder, ".txt", 1
  ]; // AbsoluteTiming

memoryCleanupStart[]
ClearAll /@ Unevaluated[{imagesFilteredCompressed, particleMasksCompressed}];
memoryCleanupFinalize[]
```

>> Import Compressed Filtered Images <<

In[∘]:=
```
filteredImagesFolder = output <> "2 Filtered Images Compressed";
launchNukes@useKernels;
SetDirectory@filteredImagesFolder;

files = SortBy[
   FileNames[],
   ToExpression@FileBaseName@# &
  ];

monitorCounter = 0;

Monitor[
 imagesFiltered = ParallelMap[
    (monitorCounter++; Uncompress@Import[
        filteredImagesFolder <> $PathnameSeparator <> #,
        IncludeMetaInformation → None
       ]) &, files
   ];, progress[monitorCounter, Length@files]
]
```

In[∘]:=
```
Manipulate[
 viewImage@rotateImage@(*ColorToneMapping@*)ImageAdjust@imagesFiltered[[k]],
 {k, 1, Length@imagesFiltered, 1}
]
```

>> Import Compressed Particle Masks <<

In[∘]:=
```
particleMasksFolder = output <> "3 Particle Masks Compressed";
launchNukes@useKernels;
SetDirectory@particleMasksFolder;

files = SortBy[
   FileNames[],
   ToExpression@FileBaseName@# &
  ];

monitorCounter = 0;

Monitor[
 particleMasks = ParallelMap[
    (monitorCounter++; Uncompress@Import[
        particleMasksFolder <> $PathnameSeparator <> #,
        IncludeMetaInformation → None
       ]) &, files
   ];, progress[monitorCounter, Length@files]
]
```

Particle Coordinates & Data for Tracking

Get Particle Coordinates

In[ ]:=
```
monitorCounter = 0;

Monitor[
  particleCentroids = (monitorCounter++;
        (Last /@ ComponentMeasurements[#, "Centroid"])
       ) & /@ particleMasks // Parallelize;,
  progress[monitorCounter, Length@particleMasks]
 ] // AbsoluteTiming

(*ListLinePlot[
  Length/@particleCentroids,FrameLabel→getLatexStyling/@{"t","N"}
 ]

Manipulate[
  rotateImage@HighlightImage[#,Style[particleCentroids[[k]],PointSize@0.005]
       ]&@getMaskOverlays[#,channelWallMask,White,2]&@imagesFiltered[[k]],
  {k,1,Length@particleCentroids,1}
 ]*)
```

<< Export Particle Coordinates >>

In[ ]:=
```
Export[
dataCompressedfolder <> $PathnameSeparator <> "Particle Centroids.txt",
Compress@particleCentroids
];
```

>> Import Particle Coordinates <<

```
particleCentroids = Uncompress@Import[
    dataCompressedfolder <> "Particle Centroids.txt"
   ];
```

Export Notebook With Parameters

In[ ]:=
```
notebookID = 1;

(*FrontEndTokenExecute["DeleteGeneratedCells"];*)
NotebookSave[];
notebookSnapshot = NotebookOpen@NotebookFileName[];

Export[
  output <> Last@StringSplit[#, $PathnameSeparator] &@output <>
   "_Run_" <> ToString@notebookID <> ".pdf", notebookSnapshot
 ];
```