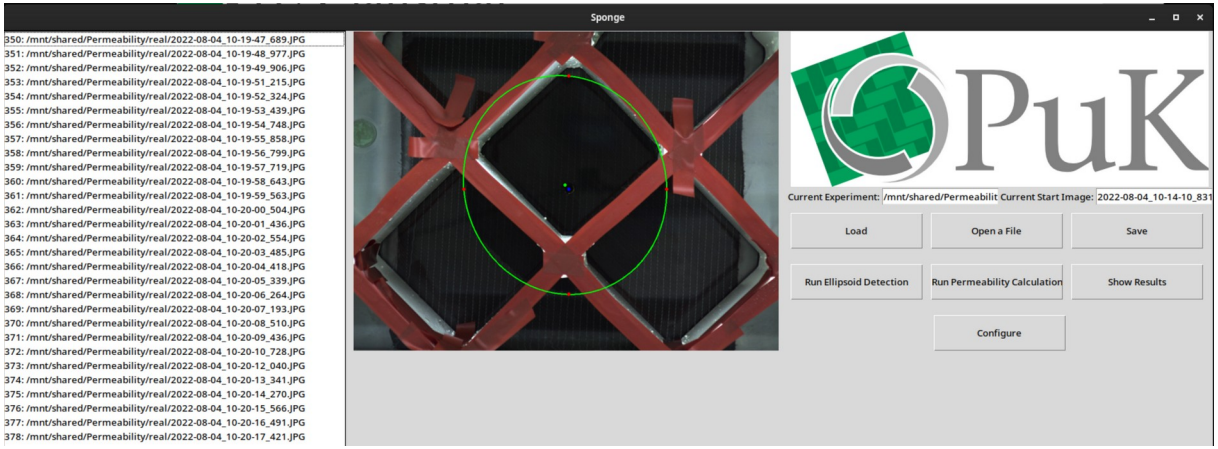
TU Clausthal

Permeability Measurements Using Image Recognition in Python

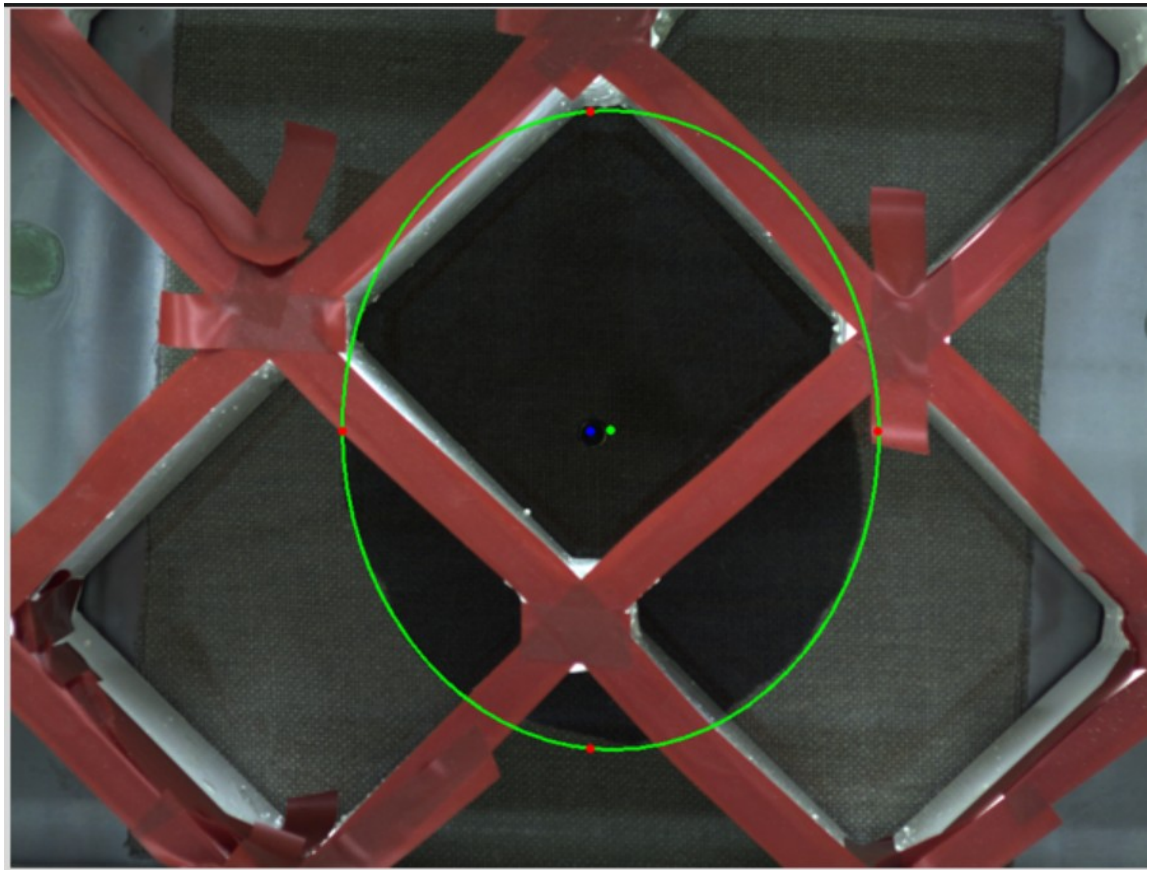
Emre Cem Elevis 500335, Informatik Student

30.08.2023



Short Overview

- An algorithm that uses a series of measurements and images that efficiently computes the general permeability of a sample.



Current Setup and its Deficiencies

Current Setup and its Deficiencies

- It only works on a specific very outdated machine.
- Space and Computation complexity increases linearly with the size of the image files, which causes higher size samples to be RAM bound and crash.
- Little to no documentation and very unfriendly development environment due to unorganized and uncommented code base and language choice. (Mix of Matlab scripts and C)

Challenges

- Sample view is obstructed due to unavoidable structural elements of the machine.
- Some frames contain unusable data, such frames should not be in the calculation.
- Part of the algorithm responsible for ellipsoid detection gets exponentially more complex with input pixels.
- Large fluctuation and variances due to ellipse approximations in the resulting data set should be avoided to achieve higher accuracy.

General Permeability Formula

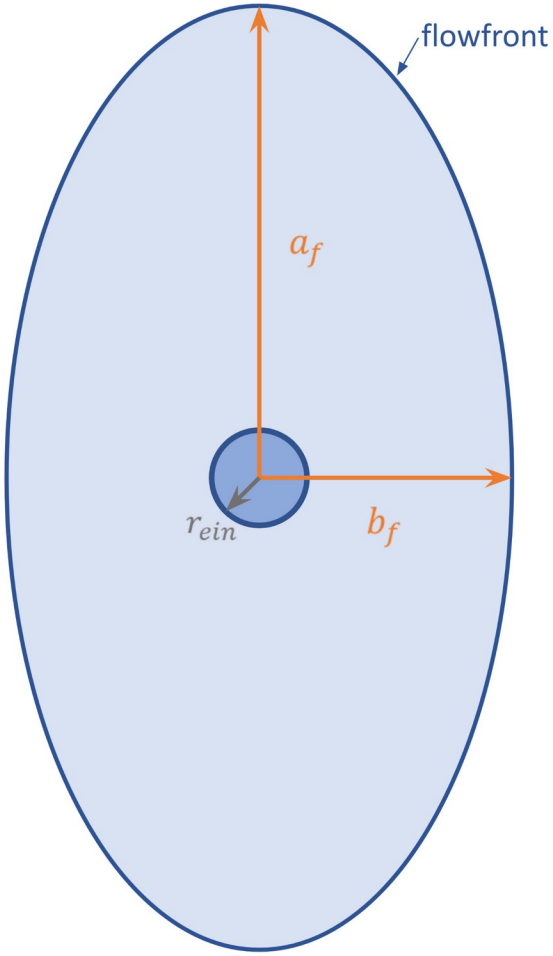
$$K_a = \frac{\Phi\eta}{4t(p_{Ff}-p_{ein})}\left(\left(a_f\sqrt{\frac{b_f}{a_f}}\right)^2\left(2ln\left(\frac{a_f\sqrt{\frac{b_f}{a_f}}}{r_{ein}}\right)-1\right)+r_{ein}^2\right)\frac{a_f}{b_f}$$

permeability in direction „a“

$$K_b = \frac{\Phi\eta}{4t(p_{Ff}-p_{ein})}\left(\left(b_f\sqrt{\frac{a_f}{b_f}}\right)^2\left(2ln\left(\frac{b_f\sqrt{\frac{a_f}{b_f}}}{r_{ein}}\right)-1\right)+r_{ein}^2\right)\frac{b_f}{a_f}$$

permeability in direction „b“

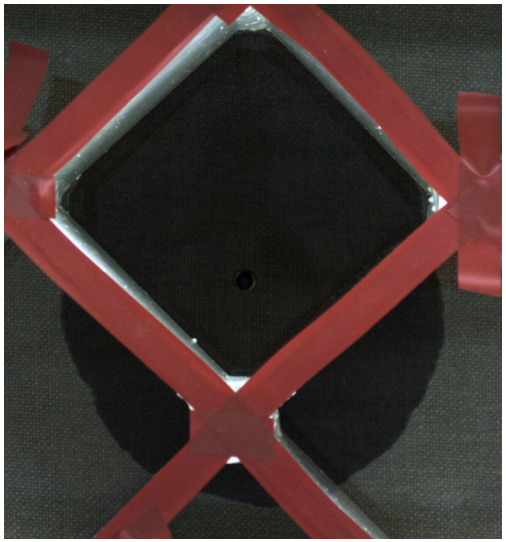
- ϕ: Porosity (can be calculated from height of textile sample, number of layers, grammage of textile, density of fiber material)
- η: viscosity of fluid used for measurement
- p_{Ff}: pressure at the flowfront
- p_{ein}: pressure at fluid inlet
- t: time



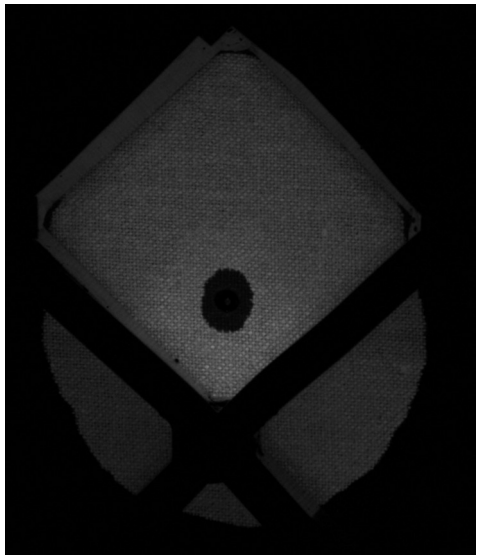
Pipeline

1) Calibration
2) Comparison of frames to mark the progression of the flow front
3) Image operations to clean the frames and achieve minimum border for ellipse generation
4) Ellipse generation using difference pixels.
5) Algebraic operations to find intersections
6) Clean up of data. (Currently using simple Interquartile range to remove outliers)
7) Solution to the permeability equation and representation of data.

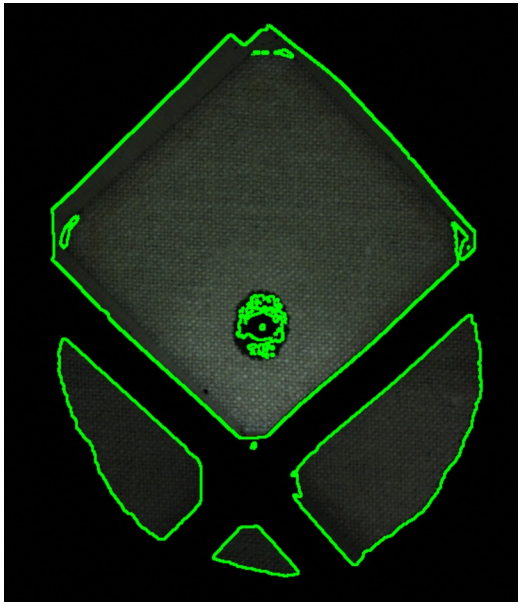
Need for Calibration



Current Frame



Difference between the first frame

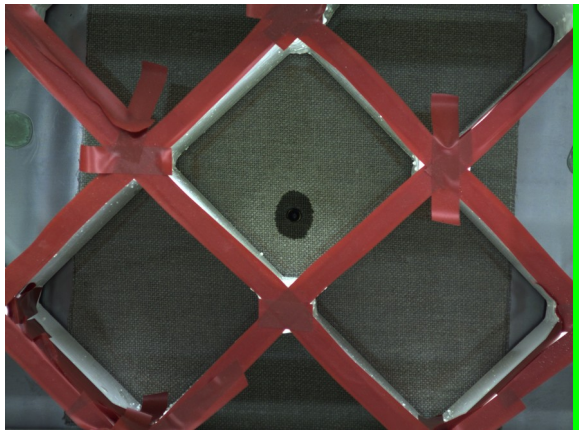


edges made by thresholding difference image

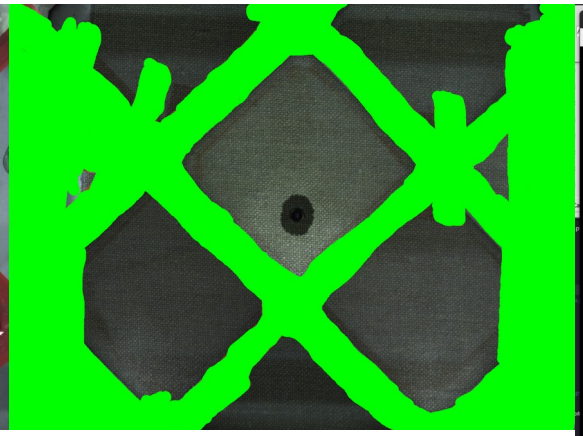
- Even with the steps mentioned in the pipeline, mitigating this unwanted reflections, a lot of effort must be spent to find the correct threshold and colour values. And even then, with dark samples such as above, correct threshold values and configs might exceed the capabilities of the camera.
- So a static mask is needed to avoid this behaviour.

Pipeline - Calibration

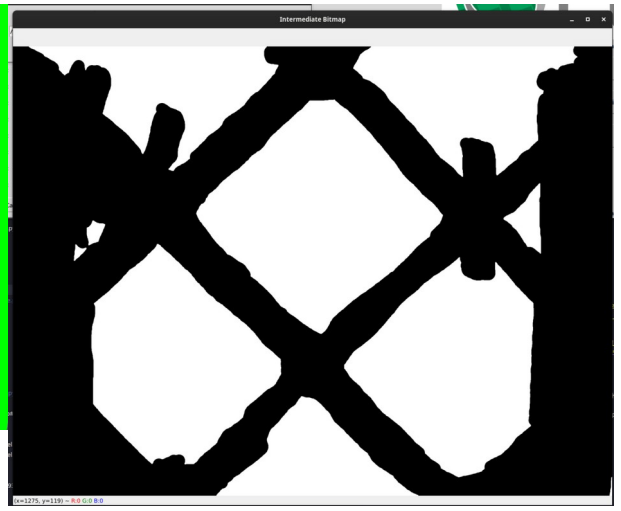
- Calibration happens in two functions, image.calibrate and image.calibrate_inverted, they are used to create masks that include and exclude the detected areas respectively. To remove noise and small unwanted regions, a morphological opening operation is applied on the mask.



Original image



calibration image „green“

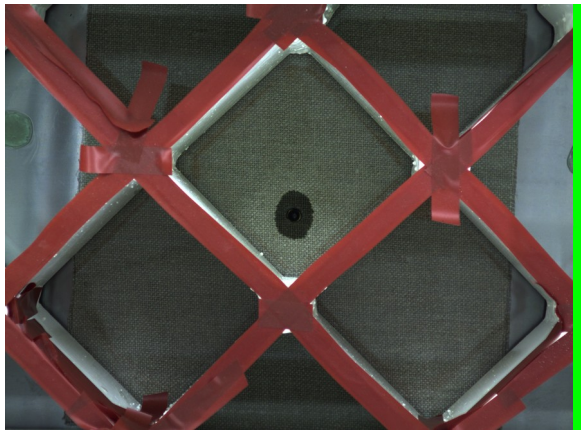


Resulting mask

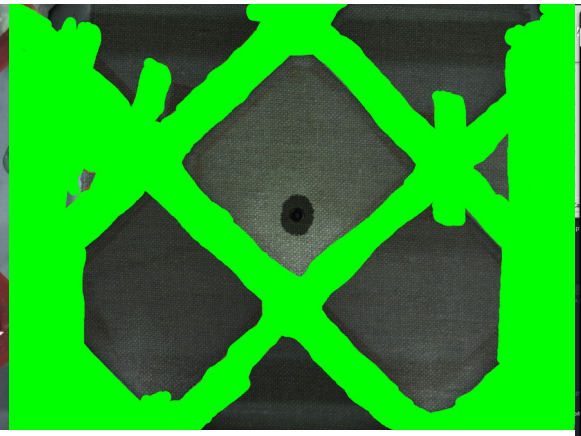
Pipeline - Calibration

- is designed to create a binary mask for an image based on a specified colour.
- It allows multiple colour calibrations, and at the end, combines all these individual masks to produce one final mask (bitmap).

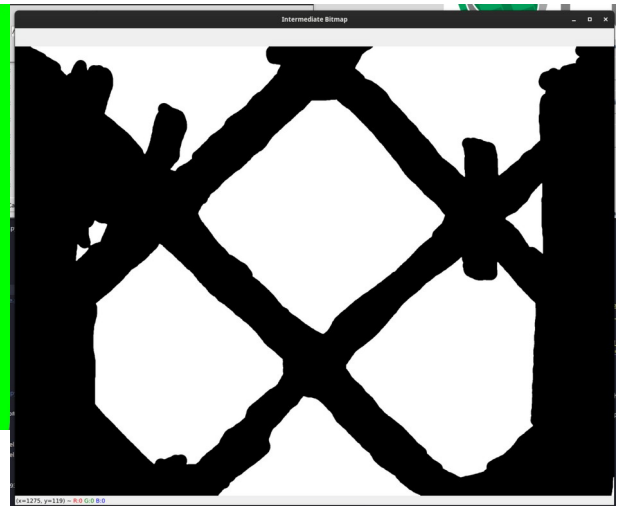
- Input list „image_data“ that has tuples of
  - path: Path to an image.
  - colour: A target colour (in normalized RGB)
  - margin: Tolerance value for colour matching



Original image

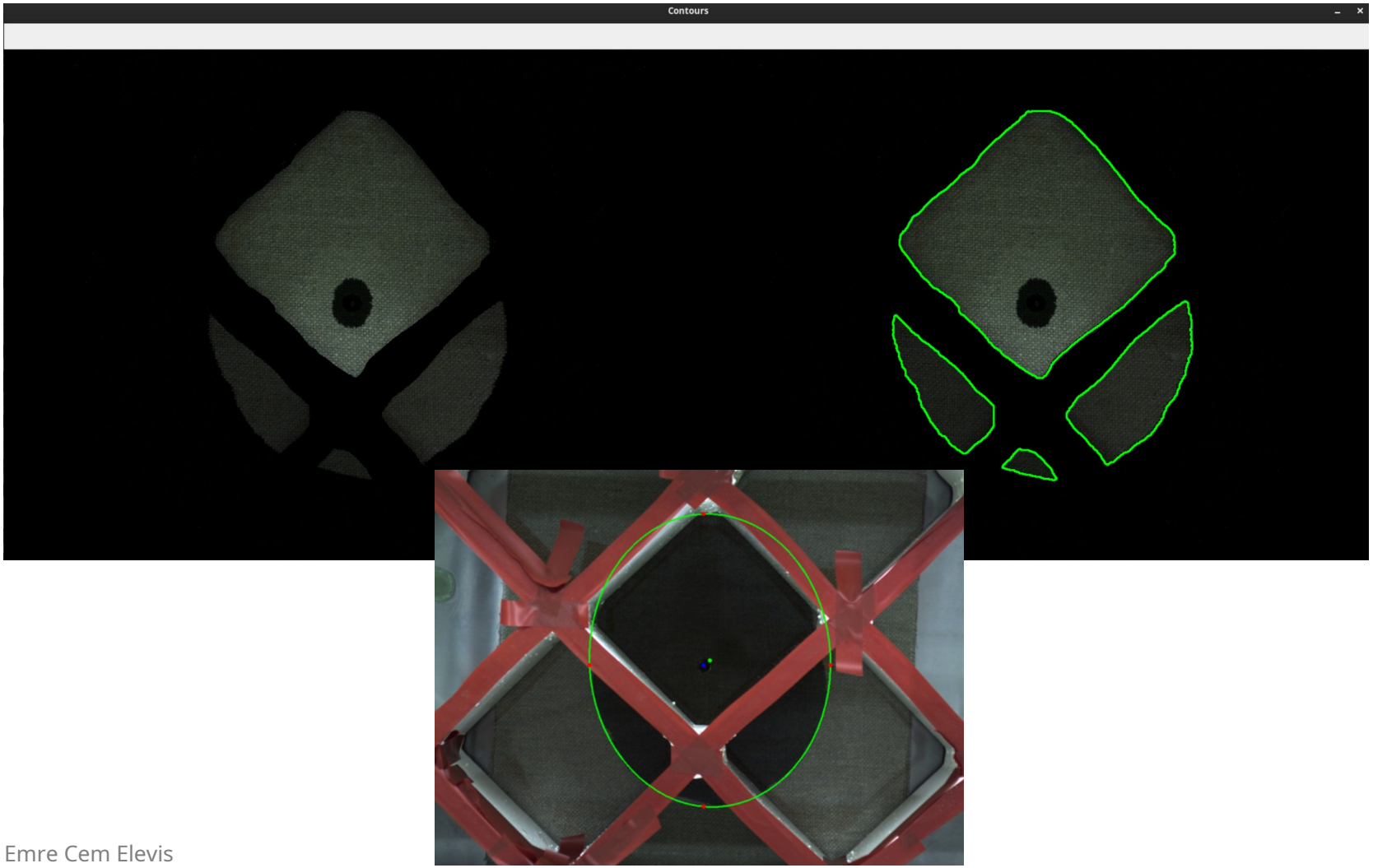


calibration image „green“



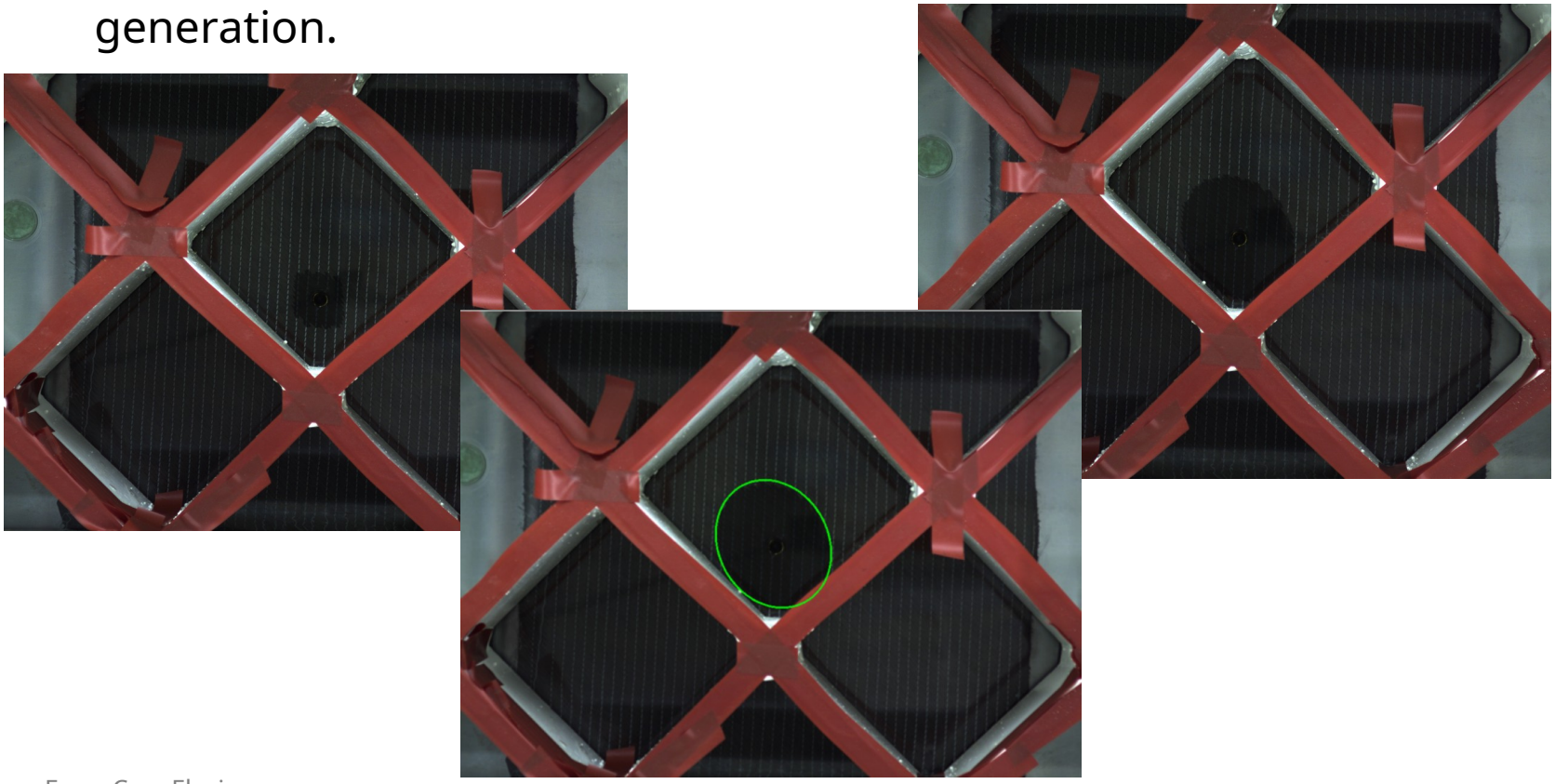
Resulting mask

Results with calibration



Pipeline – Comparison of Frames

- Comparison happens in image.subtract, is designed to compute the difference between two images, then process and visualize this difference. Eventually to achieve minimum border for ellipse generation.



Pipeline – Comparison of Frames

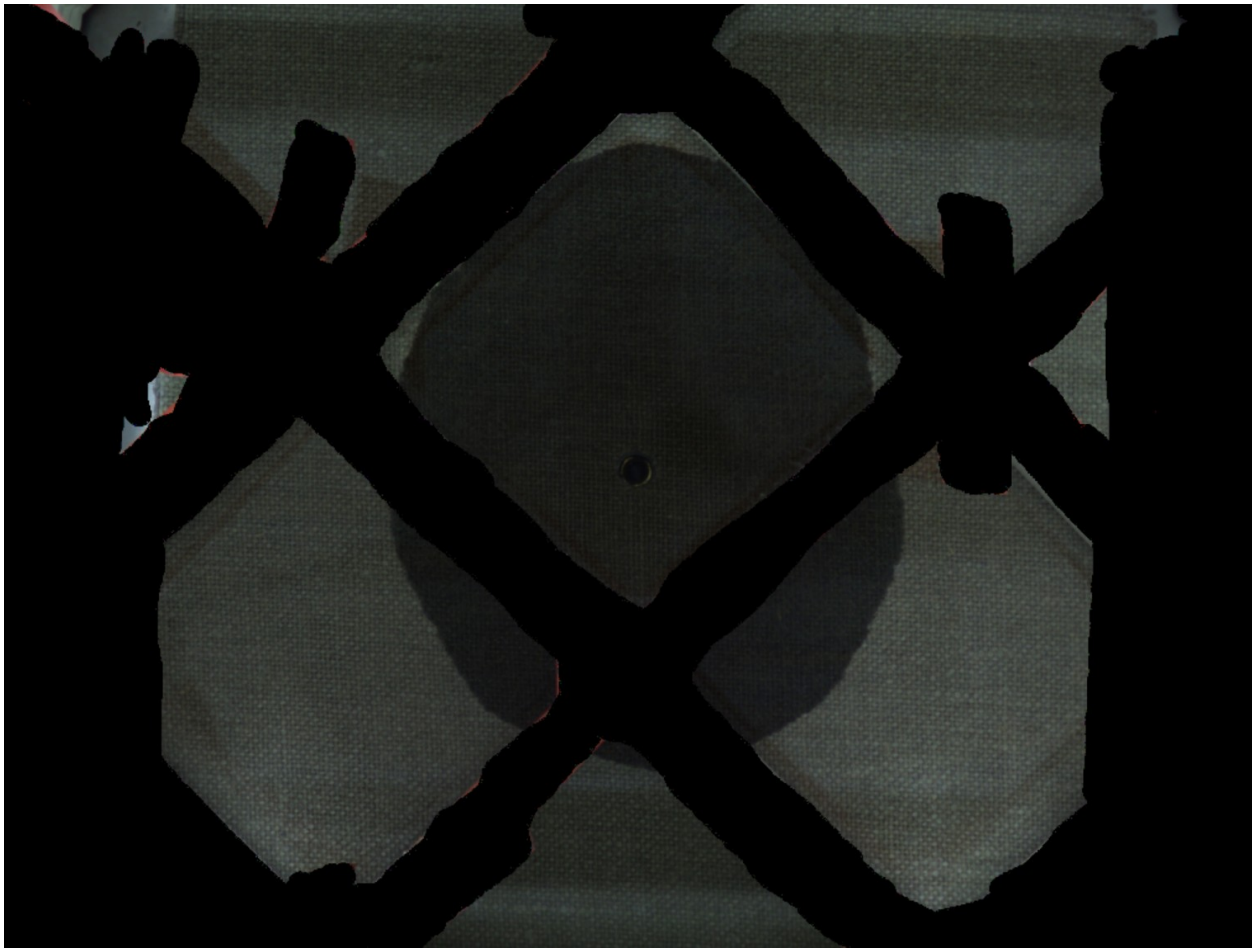
- Parameters:
  - image1, image2: Paths to two images you wish to subtract.
  - resolution: The desired output resolution for resizing intermediate results.
  - block size: Size of smallest block for morphological operations.
  - threshold: Threshold value for binary thresholding of the difference image.
  - canny_lower_thresh, canny_upper_thresh: Thresholds for the Canny edge detection.
  - mask: An optional mask for calibration
  - verbose: Boolean parameter. If set to True, displays intermediate steps.

Pipeline – Comparison of Frames

- Steps
  - **Read Images**: Load image1 and image2 using **cv2.imread**.
  - **Masking**: If a mask is provided, it's resized to match the shape of the images. Both images are then masked using **cv2.bitwise_and**(img1, mask)
  - **Compute Difference**: The absolute difference between the two images is computed. **cv2.absdiff** is used This image (Slide 10 left) shows where the two images differ.
  - **Gray scale & Blur**: Convert the difference image to grayscale **cv2.cvtColor**(diff, cv2.COLOR_BGR2GRAY) and then apply Gaussian blur **cv2.GaussianBlur**(diff_gray, (5, 5), 0). This helps in reducing noise and making thresholding more effective.
  - **Thresholding**: Binary thresholding **cv2.threshold** is applied to the blurred grayscale image. Pixels with intensities above the given threshold will be set to 255, rest to 0.
  - **Morphological Opening**: A morphological opening operation **cv2.morphologyEx** is applied to remove small noise and clusters. The size of the structuring element **cv2.getStructuringElement** is defined by block size. Removing bit groups smaller than blocksizeXblocksize from the image.
  - **Edge Detection**: Canny edge detection **cv2.Canny** is applied to detect edges in the opened image.
  - **Find Contours**: Contours of the detected edges are found **cv2.findContours**. If no contours are detected, an exception is raised. While this provides no new information, it is useful to lower the amount of points handed to the ellipse detection by removing the inner edges .

Pipeline – Comparison of Frames - Masking

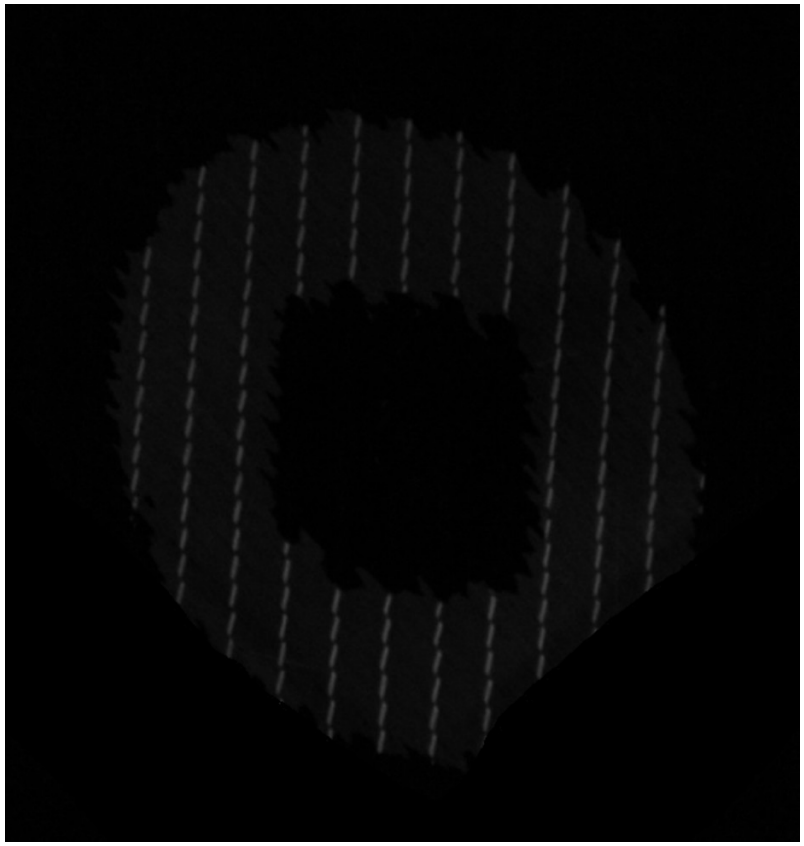
Pipeline – Comparison of Frames - Masking

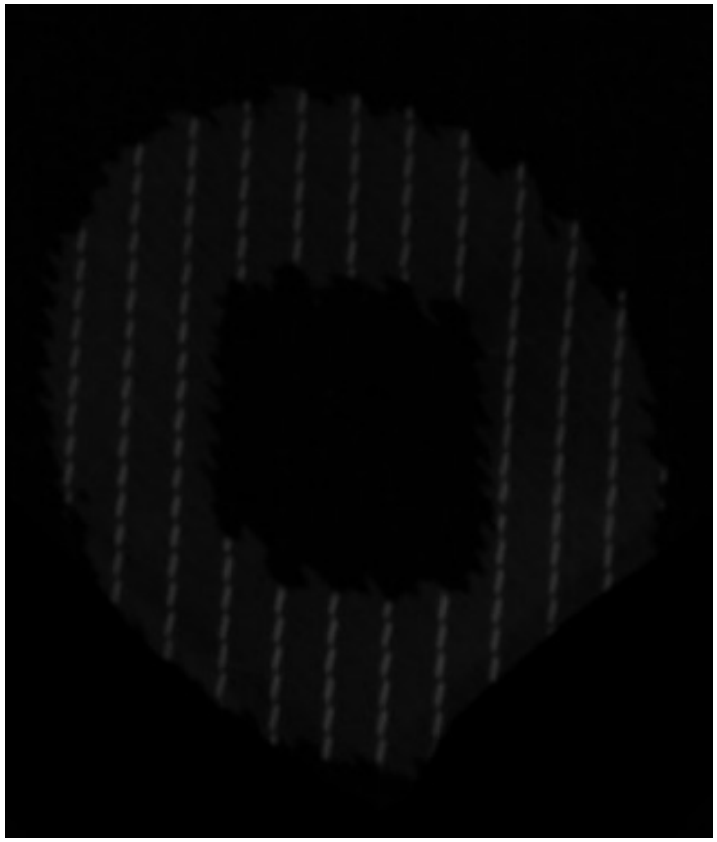


Pipeline – Comparison of Frames – Compute Difference

Pipeline – Comparison of Frames – Compute Difference

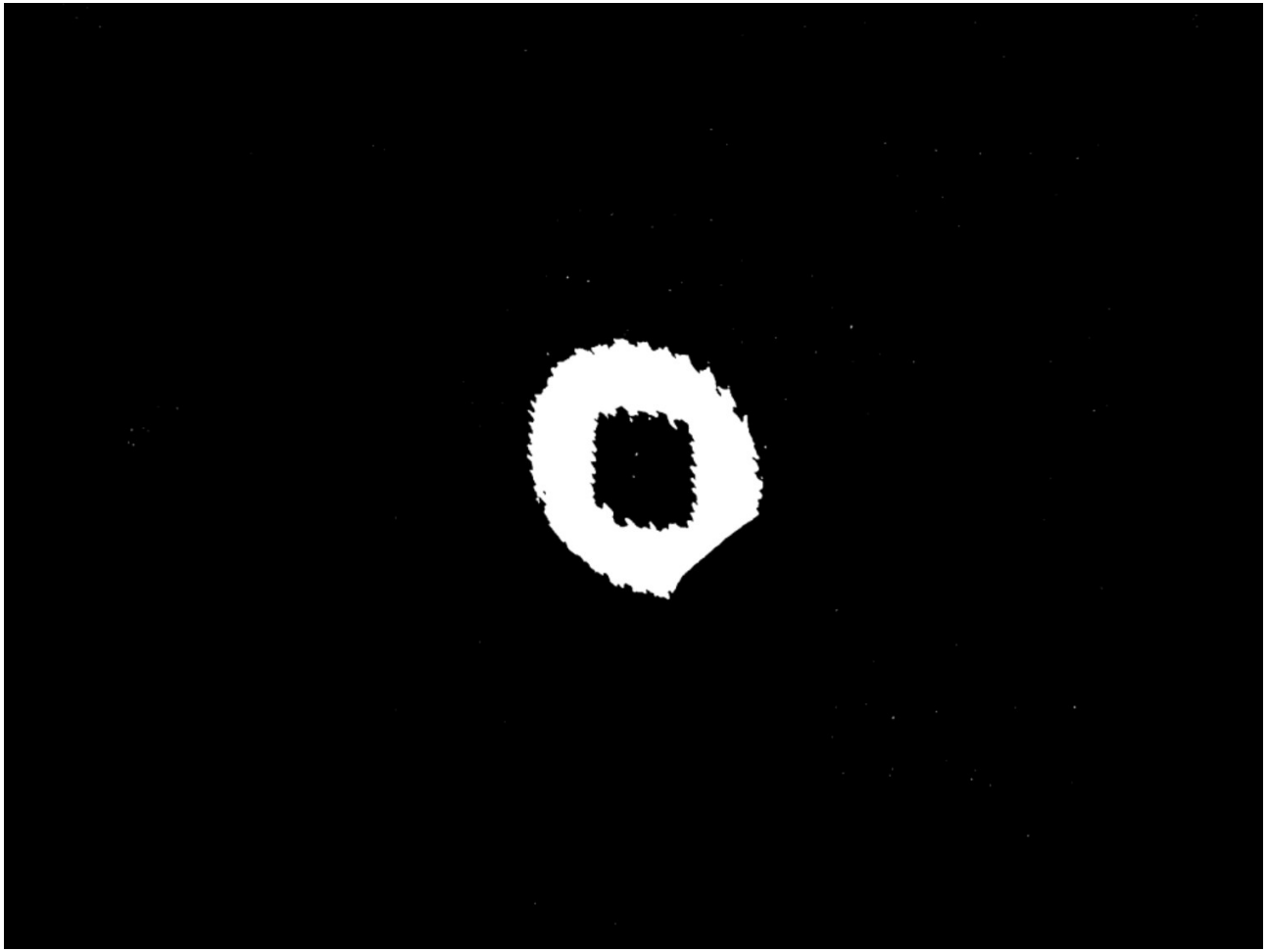
Pipeline – Comparison of Frames - Gaussian blur



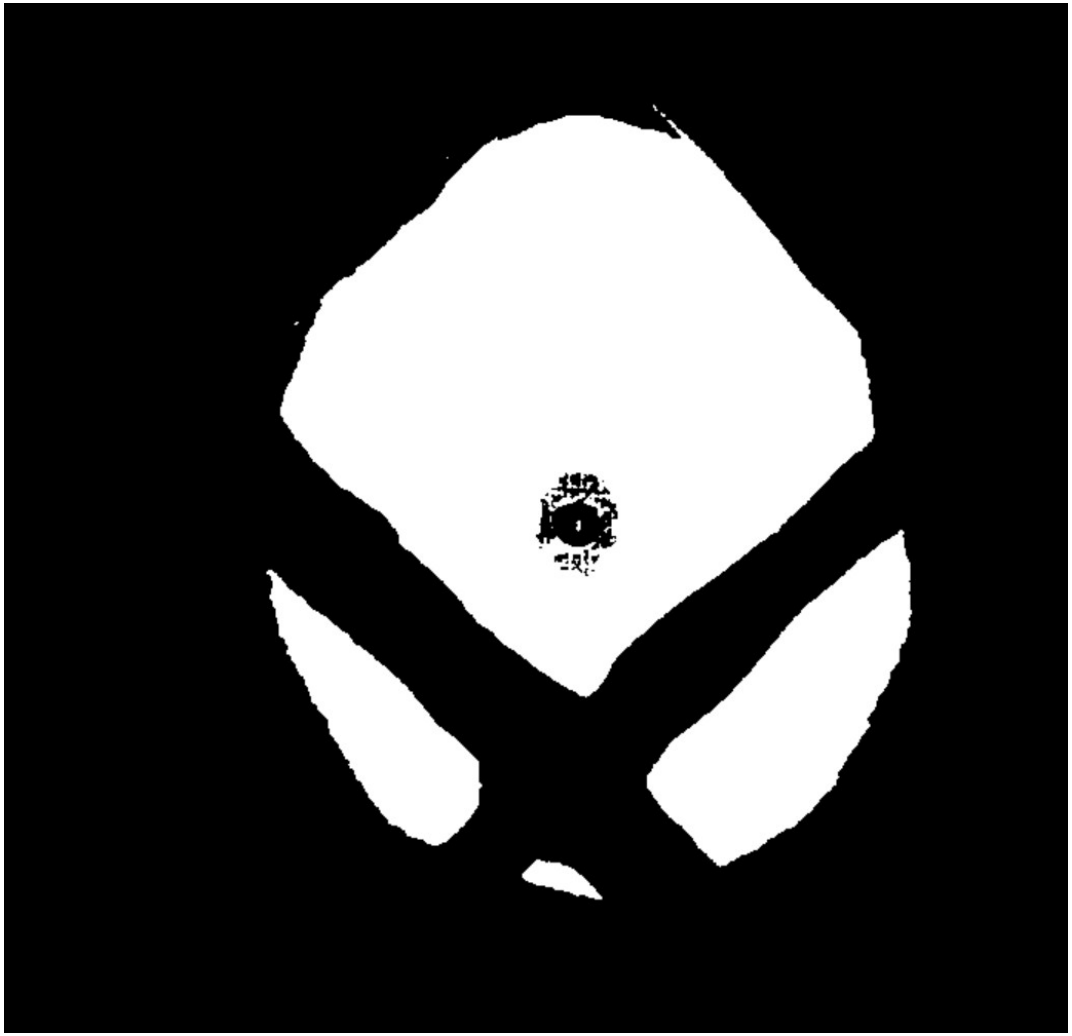


Pipeline – Comparison of Frames - Gaussian blur

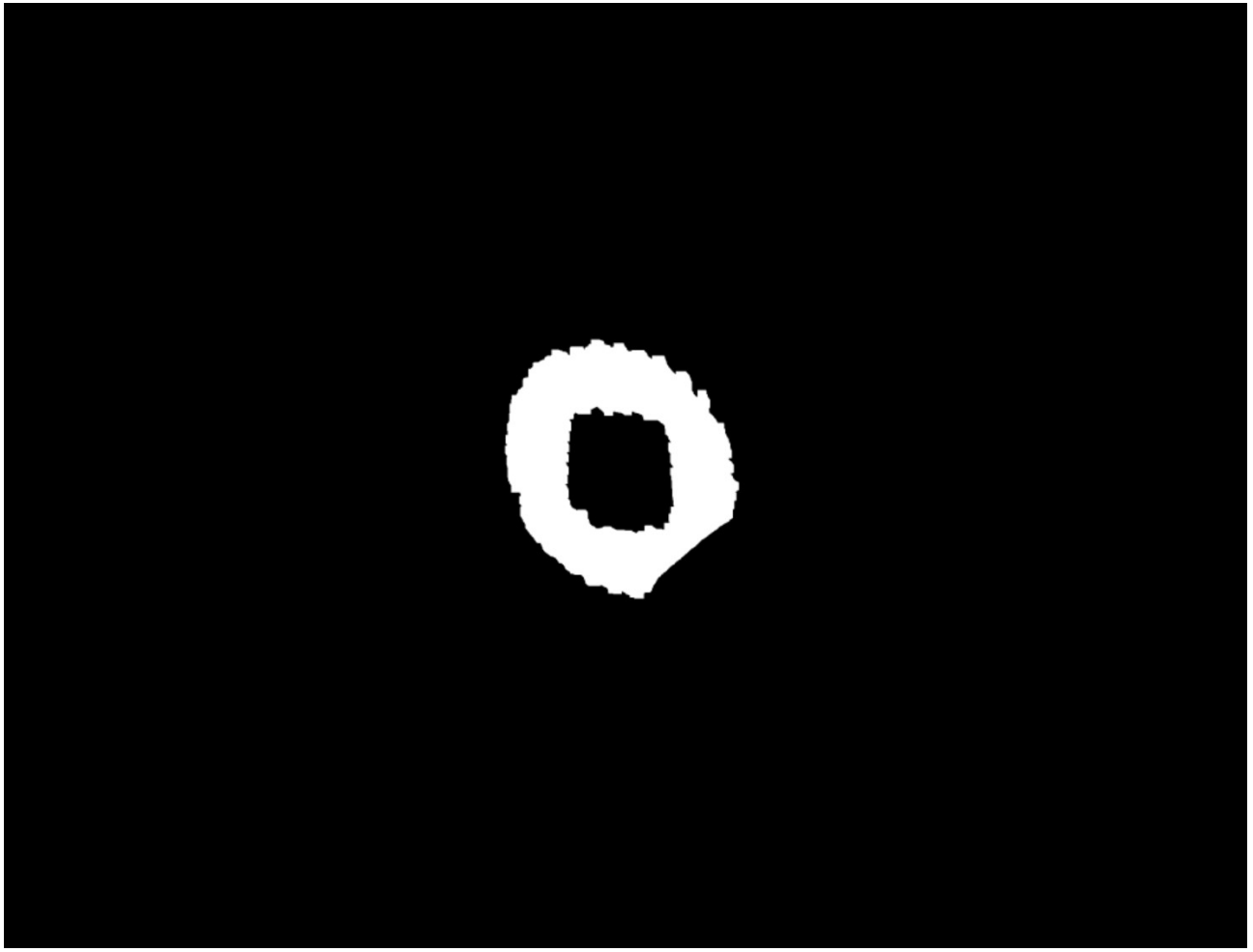
Pipeline – Comparison of Frames – Thresholding



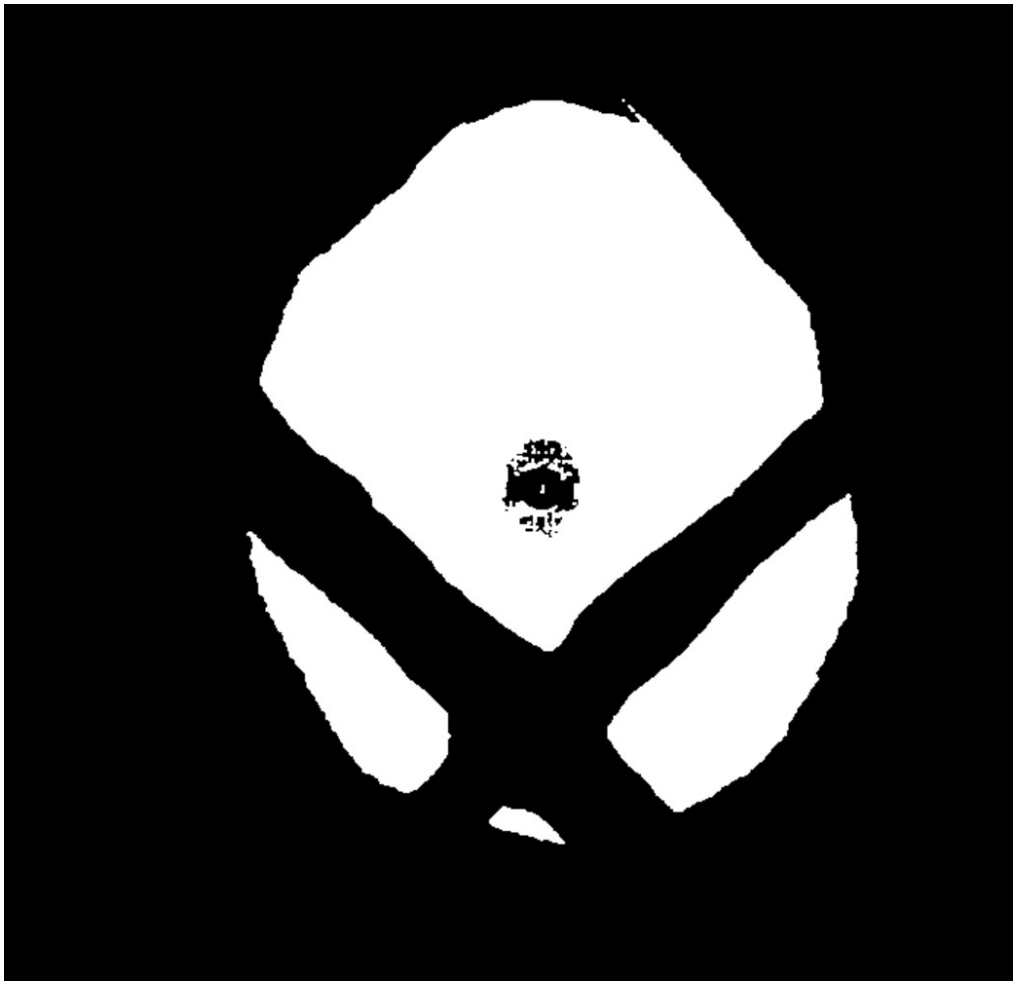
Pipeline – Comparison of Frames – Thresholding



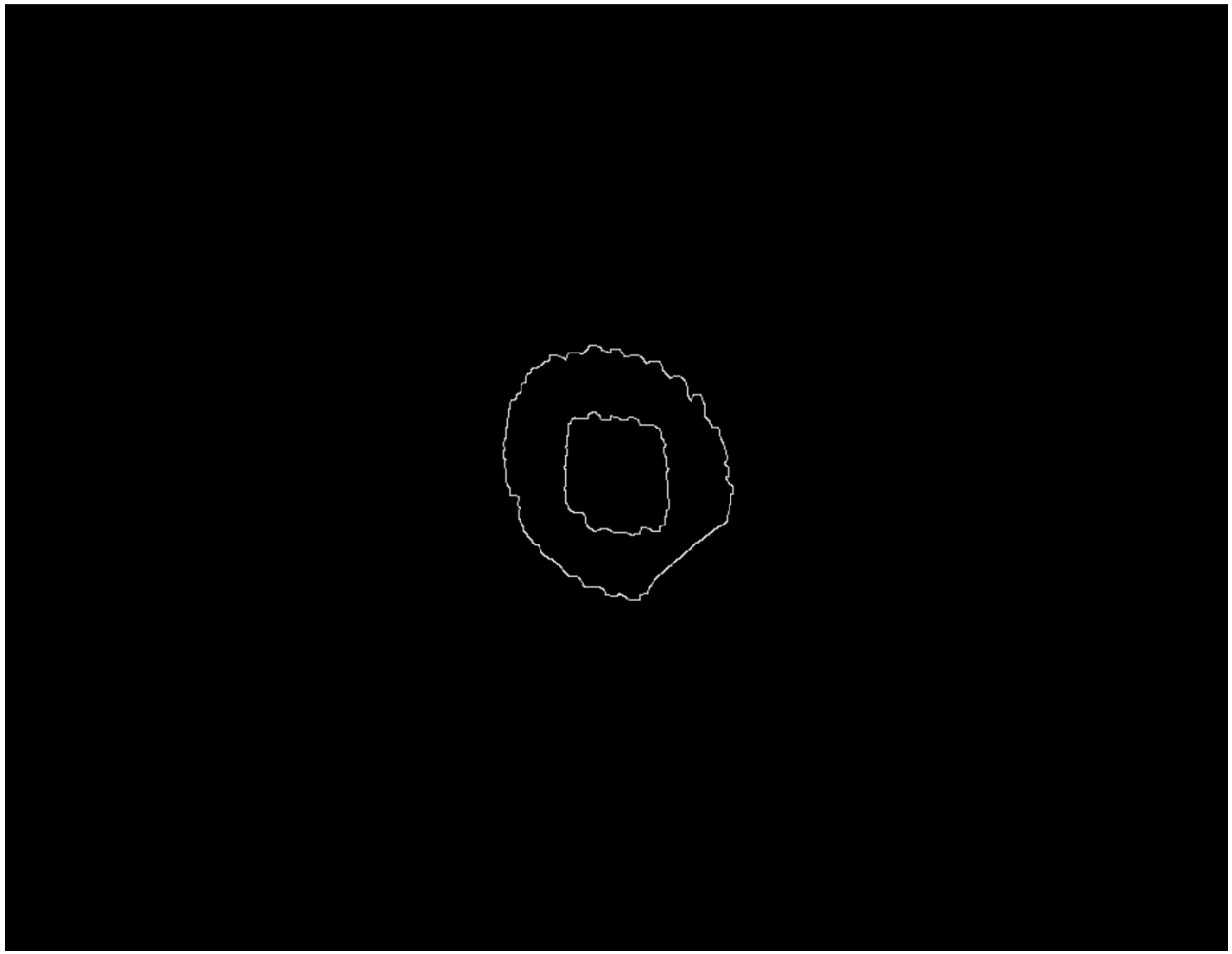
Pipeline – Comparison of Frames - Morphological Opening



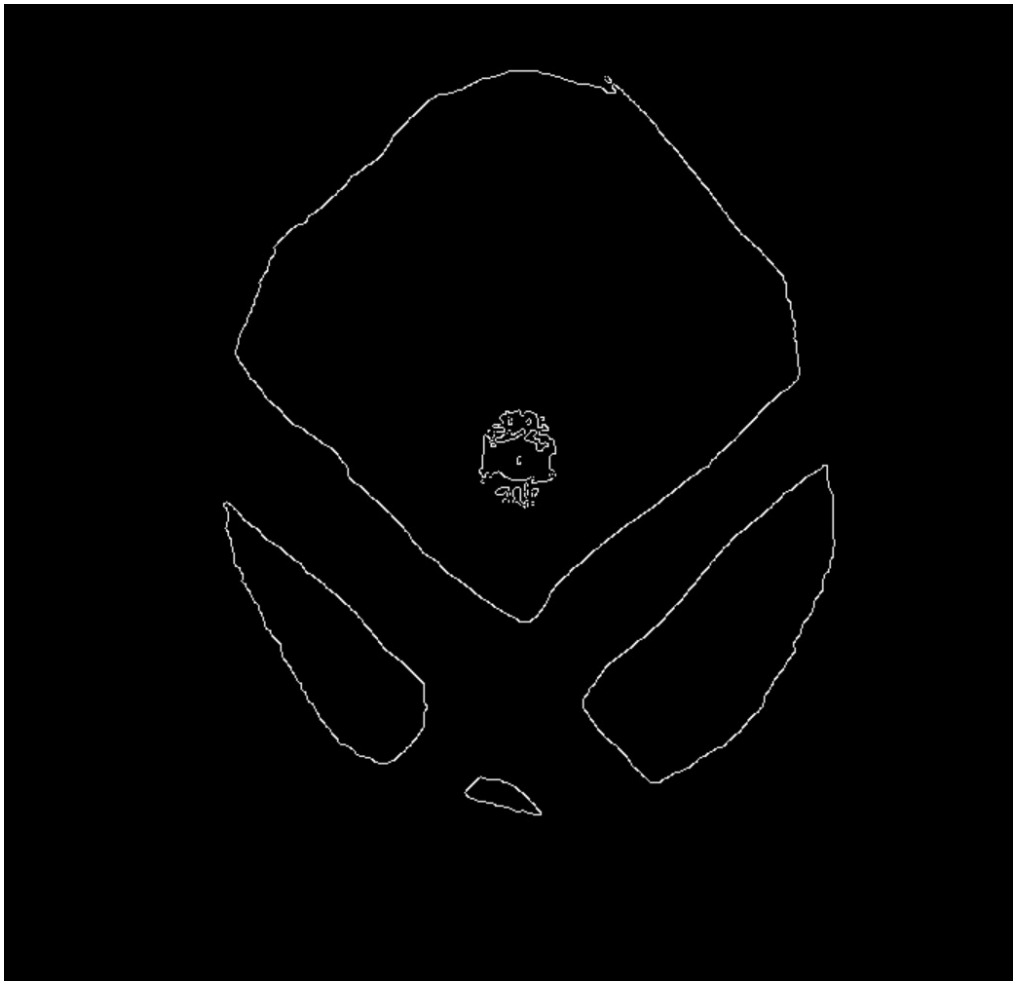
Pipeline – Comparison of Frames - Morphological Opening



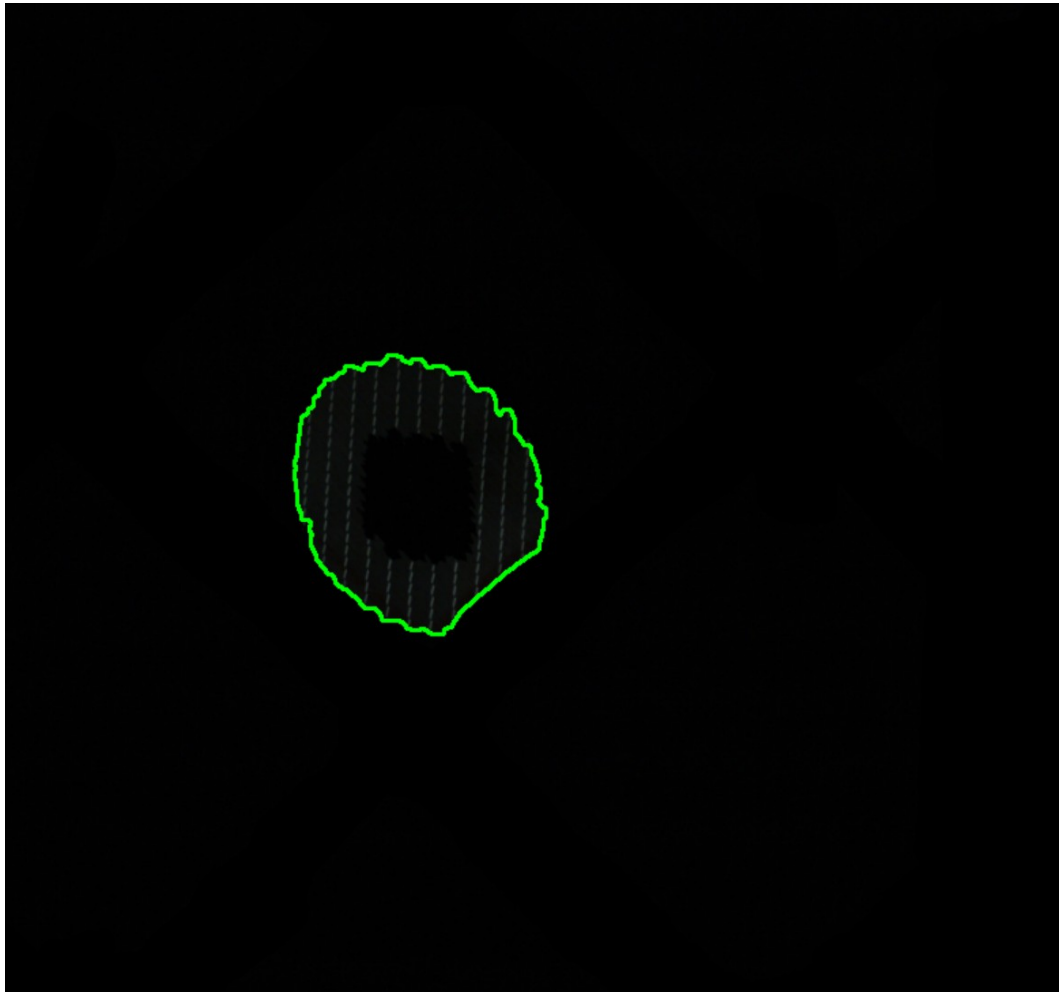
Pipeline – Comparison of Frames – Edge detection



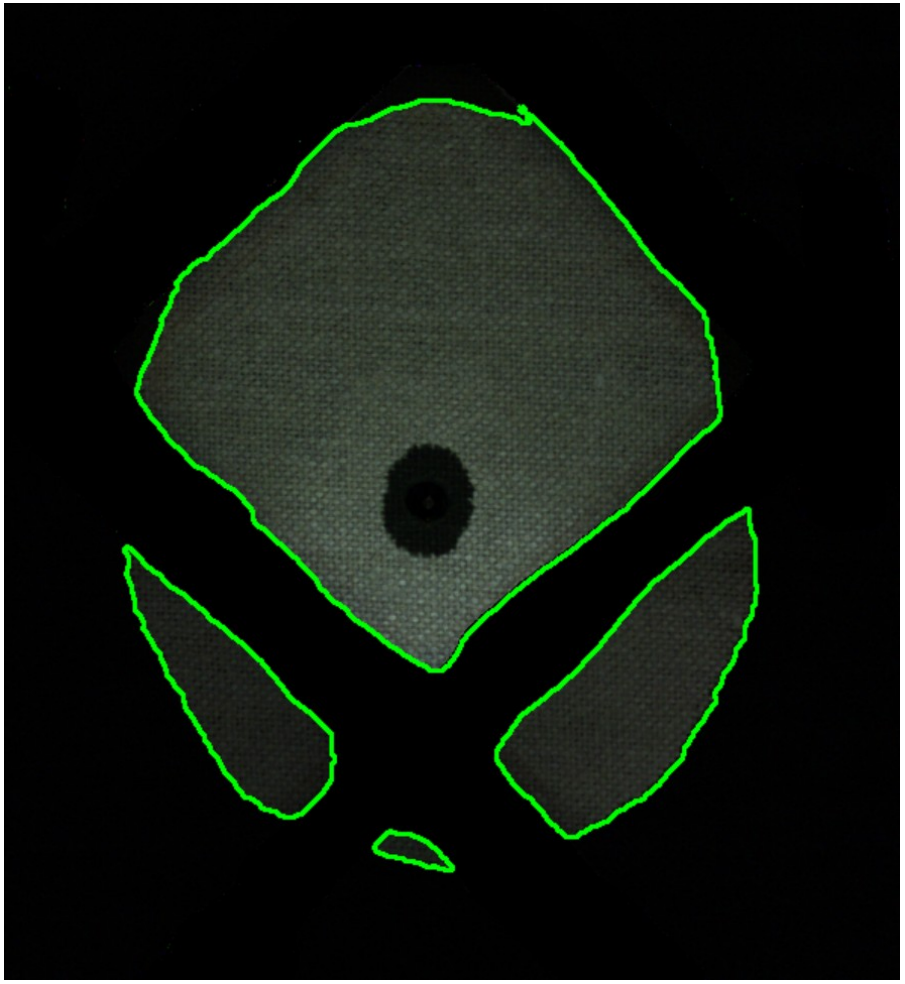
Pipeline – Comparison of Frames – Edge detection



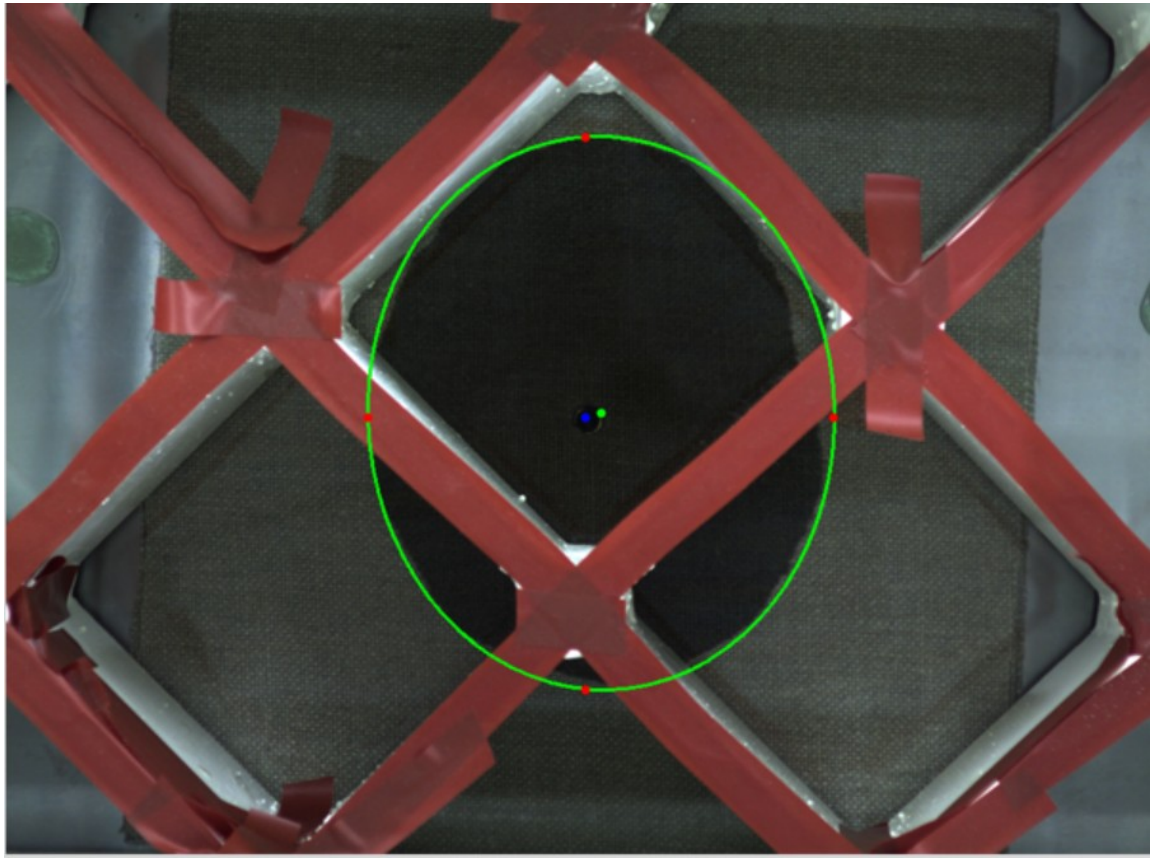
Pipeline – Comparison of Frames - Contours



Pipeline – Comparison of Frames - Contours



Pipeline – Ellipse & Flowfront Generation



Pipeline – Ellipse Generation

- Input is an numpy array of points
- Output is an array of (centre point, two radiis and rotation matrix)

```
from matplotlib import pyplot as plt
import numpy as np
from numpy import linalg

def getMinVolEllipse(P=None, tolerance=0.01):
        """ Find the minimum volume ellipsoid which holds all the points

        Based on work by Nima Moshtagh
        http://www.mathworks.com/matlabcentral/fileexchange/9542
        and also by looking at:
        http://cctbx.sourceforge.net/current/python/scitbx.math.minimum_covering_ellipsoid.html
        Which is based on the first reference anyway!

        Here, P is a numpy array of N dimensional points like this:
        P = [[x,y,z,...], <-- one point per line
             [x,y,z,...],
             [x,y,z,...]]

        Returns:
        (center, radii, rotation)

        """
```

- Source code

Pipeline – Algebraic operations

- We are using the sympy library to create lines that represent af and bf and solving for the intersections between the ellipse and lines to find af and bf values.

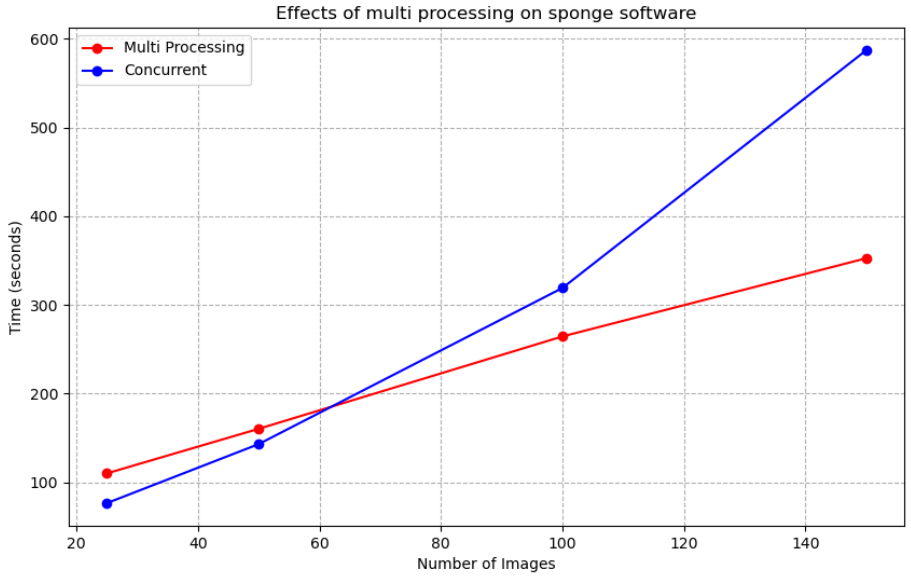
```
# Line parameters
x0, y0 = line[0]  # A point on the line
phi = np.radians(line[1])  # Angle with respect to x-axis, converted to radians

# Ellipse equation (rotated)
ellipse_eq = Eq(((x - h)*np.cos(theta) + (y - k)*np.sin(theta))**2/a**2 +
                ((x - h)*np.sin(theta) - (y - k)*np.cos(theta))**2/b**2, 1)

# Line equation
line_eq = Eq(y - y0, np.tan(phi)*(x - x0))

# Solve the system of equations
solutions = solve((ellipse_eq, line_eq), (x, y))
```

Features

- Ability to save and load the sample as well as a config file.
- Ability to remove unwanted frames.
- Ability to display all intermediary operations.
- Ability to change parameters graphically.
- Supports Linux, MacOS (tested only on intel), Windows
- Ability to graph the growth of permeability in one axis divided by the nozzle center. (Allows consistency insights for fabrics)
- Ability to load the necessary frames as they are in need to ensure low complexity.
- Ability to calibrate measuring area either with a manually edited image, or with a carton laid over the measuring area.
- Ability to compute the sample as parallel operations

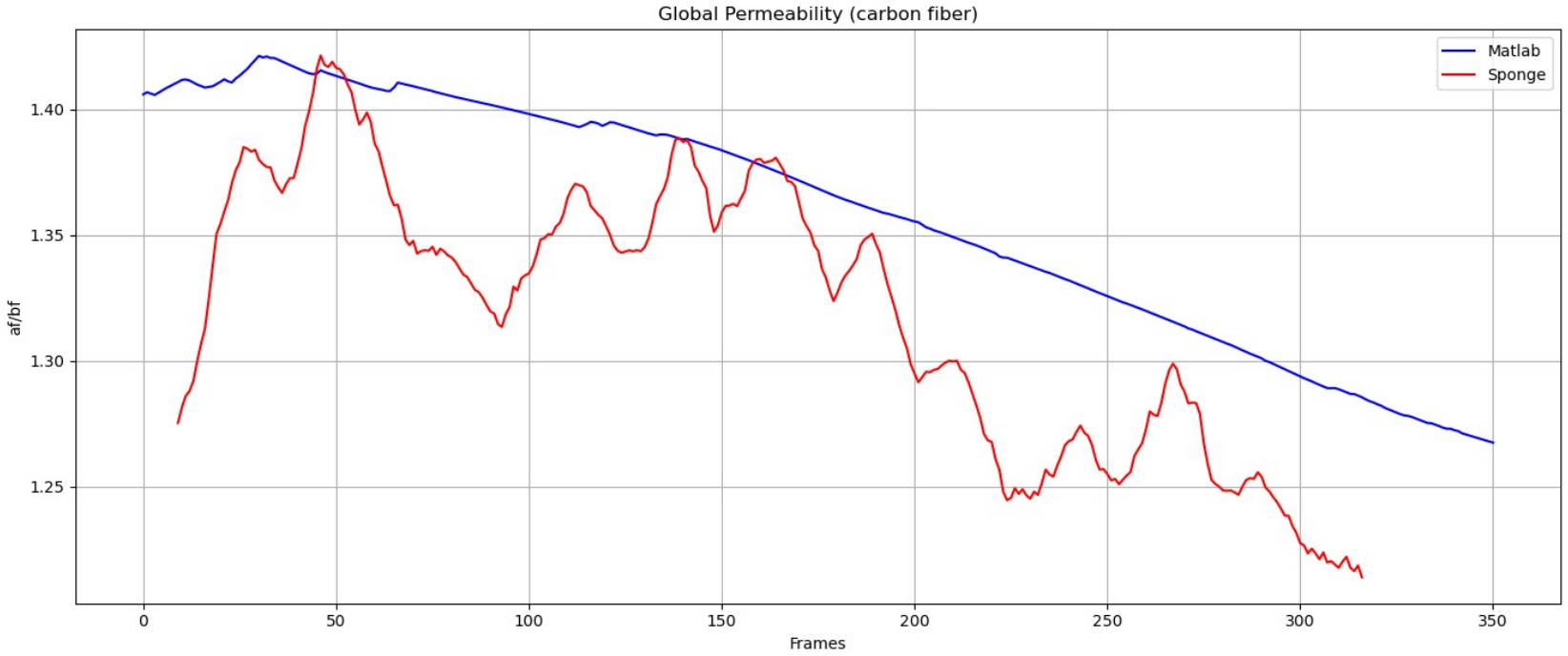
Performance Results

- Average of 3 samples with 4 different sample sizes, 70% resolution, 10 cores of ryzen 3600 4.2 ghz x 12
- The initial set-up and the allocation of the python multiprocessing module, and the subsequent handling of the said processes by the OS makes concurrent processing more feasible in smaller sample sizes



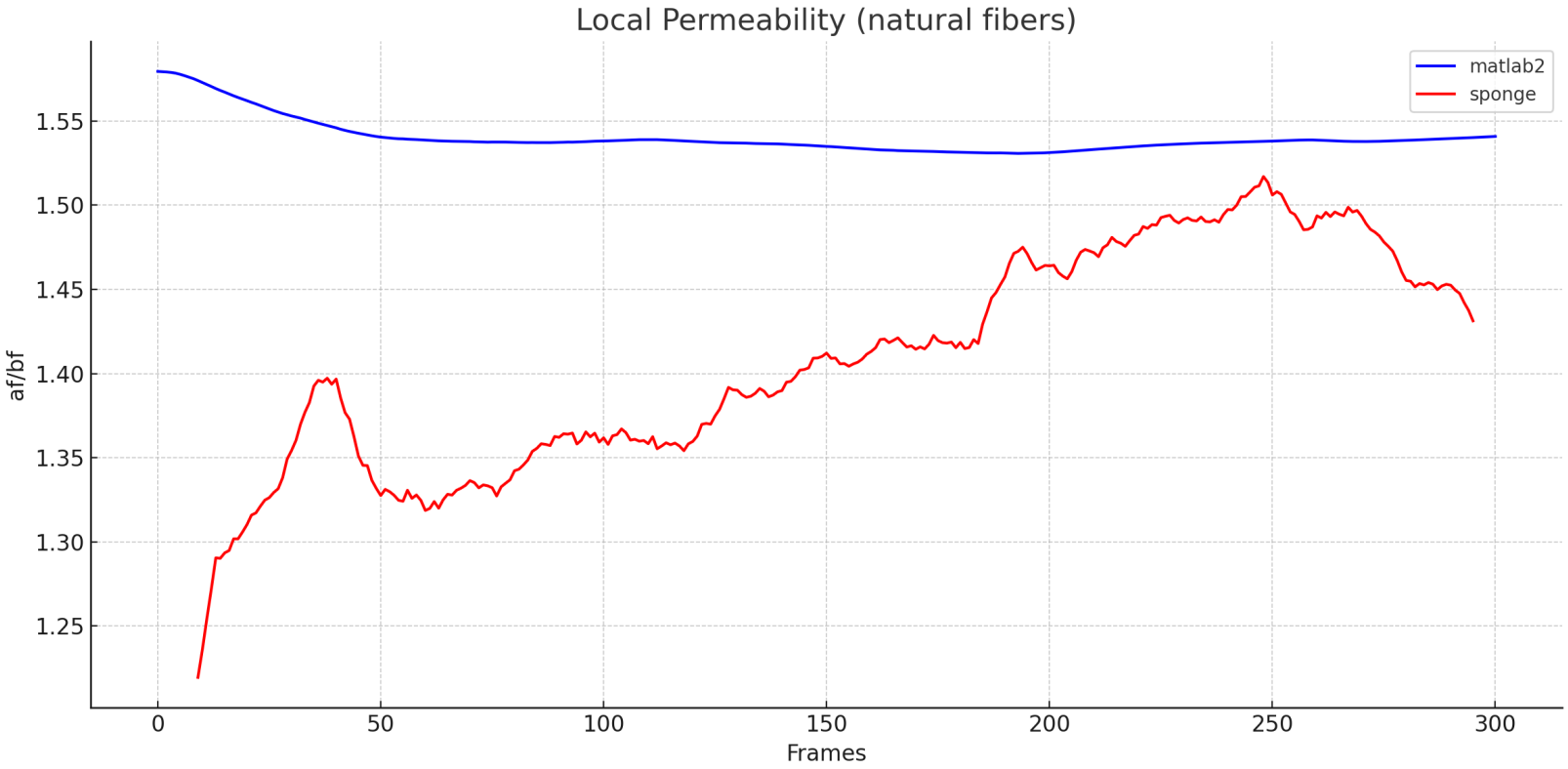
Results and Comparison

- Carbon Fiber (dark fabric) ratio of permeabilities in two axis, compared to matlab results.



Results and Comparison

- Natural Fibers (light fabric) ratio of permeabilities in two axis, compared to matlab results.



Bad Cases

- The most computationally intense part of the calculation is the ellipse creation step. Which exponentially becomes more complex with the inputted pixels. The countermeasures taken, do not account for unrealistic changes in the experiment (shadow of a researcher passing, or oil spilling) This causes considerably higher wait times when compared to the previous solution which correctly detects such bad data with „confidence ellipses“.

- Darker fabrics with low contrasts need more tinkering with the settings to be made feasible.

Possible Improvements

- Using Data science methods to find optimum parameters and approach to take on when cleaning the sample image.

- Ellipse creation can be done much more efficiently if it is done incrementally. Since between frames a lot of the points are shared and the process for ellipses is recursive, after the initial calculation of an ellipse, the additional points can be added removing the need for repeated operations.

Possible Improvements

- While generating the ellipsoid create two or four mirrored ellipsoids and average them out to avoid rotation fluctuations that distort af and bf values. Mirror operation should be done on the ellipse center and according to the axis of permeability. Can be achieved computationally easier by manipulating the rotational matrices of the ellipses.

  Shows promising results on small data sets. Requires study to define when is it worth to let go of rotational data and to what extent is feasible. A method to decide when to use and to what extent can be used, for example after a statistically decided n number of samples has been processed, and if the angle of ellipse has varied greatly between those frames. This operation which normally should lower accuracy due to loss of detail might prove useful then.

Demonstration

REFERENCES

- https://github.com/jhu-asco/ellipsoid_package/blob/master/ellipsoid_tool.py
- http://cctbx.sourceforge.net/current/python/scitbx.math.minimum_covering_ellipsoid.html
- K.L. Adams, W.B. Russel, L. Rebenfeld, Radial penetration of a viscous liquid into a planar anisotropic porous medium

- Thank you for your time and attention.