Digital Image Processing

Daniel T. McGuiness, PhD

Version: γ.2024.5

MCI

B.Sc - Digital Image Processing

Table of Contents

Histogram Operations

Table of Contents

Histogram Operations

- Frequently, an image is scanned in such a way that the resulting brightness values do not make full use of the available dynamic range .
- By stretching the histogram over the available dynamic range we can attempt to correct this range under-use.
- If the image is intended to go from brightness 0 to brightness $2^B - 1$, then one generally maps the 0 value to the value 0 and the 100 value to the value $2^B 1$.
- The appropriate transformation is given by:

$$b[m, n] = \left(2^B - 1\right) \frac{a[m, n] - \text{minimum}}{\text{maximum} - \text{minimum}}.$$

- This formula, can sensitive to outliers and a less sensitive.

- A more general description is given by:

$$b[m, n] = \begin{cases} 0 & a[m, n] \leq p_{\text{lo}}\%, \\ (2^B - 1)\,\dfrac{a[m, n] - p_{\text{lo}}\%}{p_{\text{hi}}\% - p_{\text{lo}}\%} & p_{\text{lo}}\% < a[m, n] < p_{\text{hi}}\%, \\ (2^B - 1) & a[m, n] \geq p_{\text{hi}}\%. \end{cases}$$

- In this version one might choose the 1% and 99% values for $p_{\text{lo}}\%$ and $p_{\text{hi}}\%$, respectively, instead of the 0% and 100% values.
- It is also possible to apply the contrast-stretching operation on a regional basis using the histogram from a region to determine the appropriate limits for the algorithm.
- It is possible to suppress the term $2^B - 1$ and simply normalize the brightness range to $0 \leq b[m, n] \leq 1$.
- This means representing the final pixel brightnesses as reals instead of integers but modern computers can handle this with ease.

Equalisation

- Before we compare two images, it is always a good practice to normalise their histograms to a **standard** histogram.
- This is useful when images were retrieved from different sources.
- The most common technique is **histogram equalisation** where ne attempts to change the histogram through the use of a function $b = f(a)$ into a histogram that is constant for all brightness values.
- This would correspond to a brightness distribution where all values are equally probable.
- Unfortunately, for an arbitrary image, one can only approximate this result.

Equalisation

- For a suitable function $f(\cdot)$ the relation between the input probability density function, the output probability density function, and the function $f(\cdot)$ is given by:

$$p_{\mathrm{b}}(b)\ db = p_{\mathrm{a}}(a)\ da \quad \rightarrow \quad \frac{p_{\mathrm{a}}(a)\ da}{p_{\mathrm{b}}(b)}.$$

- Here we see that suitable means $f(\cdot)$ is differentiable and that df/da ≥ 0. For histogram equalization we desire that pb(b) = constant and this means that:

$$f(a) = \left(2^B - 1\right) \cdot P(a)$$

where $P(a)$ is the probability distribution function.

- In other words, the quantized probability distribution function normalized from 0 to $2^B - 1$ is the look-up table required for histogram equalization.

Histogram Operations

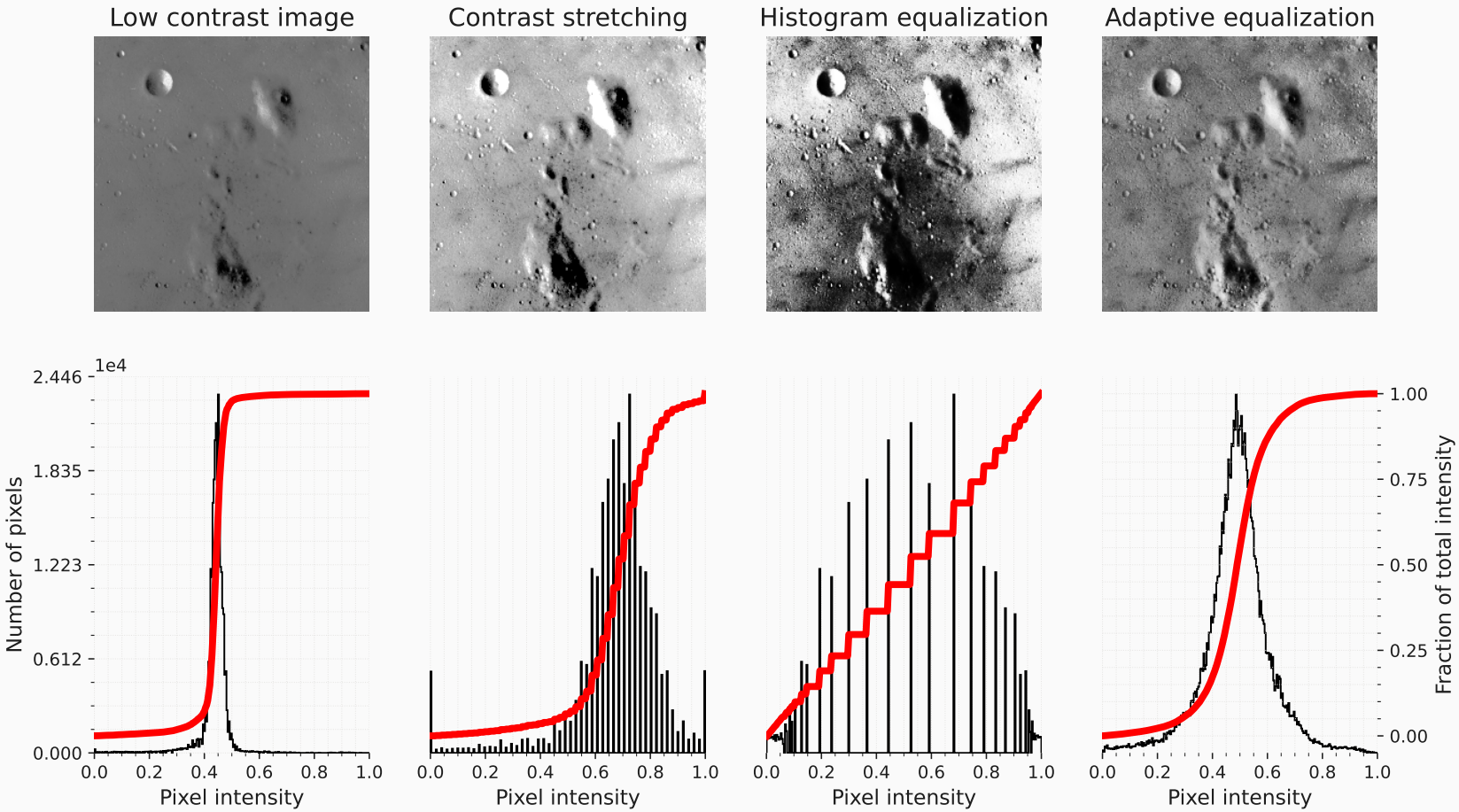


Figure 1: Different Types of contrast stretching methods.

Histogram Operations

- Thresholding is used to create a binary image from a grayscale image.
- It is the simplest way to segment objects from a background.
- This study will use the `skimage` module which offers it in two (2) ways:
  - Histogram-based. The histogram of the pixels' intensity is used and certain assumptions are made on the properties of this histogram (e.g. bimodal).
  - Local. To process a pixel, only the neighboring pixels are used. These algorithms often require more computation time.

Histogram Operations

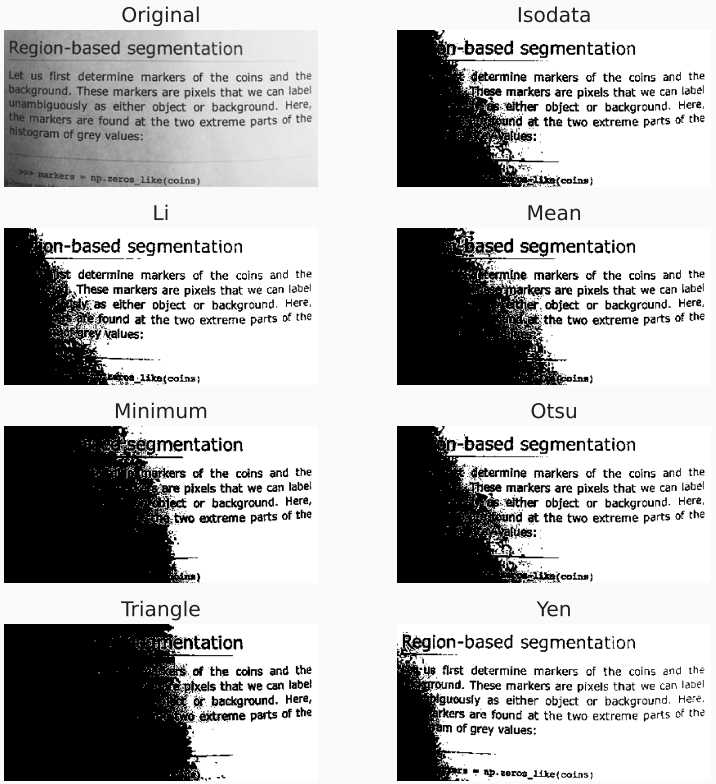


Figure 2

Histogram Operations

- illustrate how to apply one of these thresholding algorithms. This example uses the mean value of pixel intensities. It is a simple and naive threshold value, which is sometimes used as a guess value.

Histogram Operations

```
from skimage.filters import threshold_mean

image = data.camera()
thresh = threshold_mean(image)
binary = image > thresh

fig, axes = plt.subplots(ncols=2, figsize=(8, 3))
ax = axes.ravel()

ax[0].imshow(image, cmap=plt.cm.gray)
ax[0].set_title('Original image')

ax[1].imshow(binary, cmap=plt.cm.gray)
ax[1].set_title('Result')

for a in ax:
    a.set_axis_off()

cp.store_fig("threshold-mean", close=True)
```

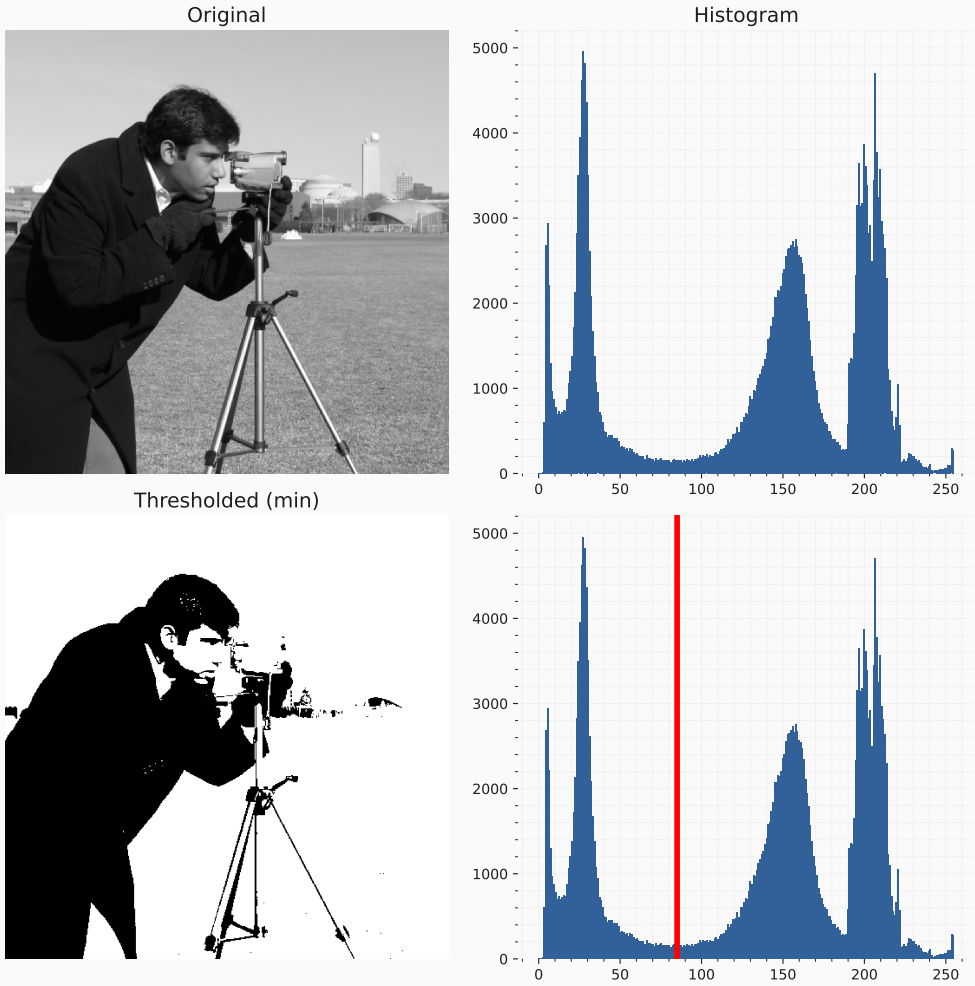

Original
Histogram
Thresholded (min)
5000
4000
3000
2000
1000
0
0
50
100
150
200
250

Figure 3

- Named after Nobuyuki Otsu, is used to perform automatic image thresholding [**sezgin2004survey**].
- In the simplest form, it returns a single intensity threshold separating pixels into two (2) classes:
  1. foreground,
  2. background,

Threshold is determined by minimizing intra-class intensity variance, or equivalently, by maximizing inter-class variance [**otsu1975threshold**].

Histogram Operations

- If background is relatively uniform, use a global threshold value.
  - such as Otsu's method.
- However, if there is large variation in the background intensity, adaptive thresholding (i.e., local or dynamic thresholding) can produce better results.

Note that local is much slower than global thresholding.

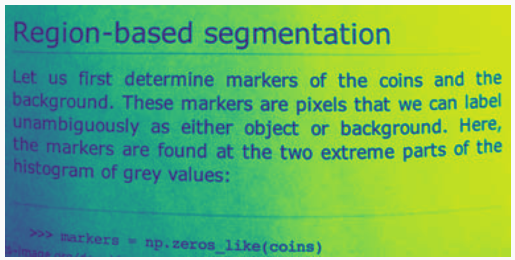
Here, we binarize the image using the `threshold_local` function, which calculates thresholds in regions with a characteristic size `block_size` surrounding each pixel. Each threshold value is the weighted mean of the local neighbourhood minus an offset value.

Histogram Operations

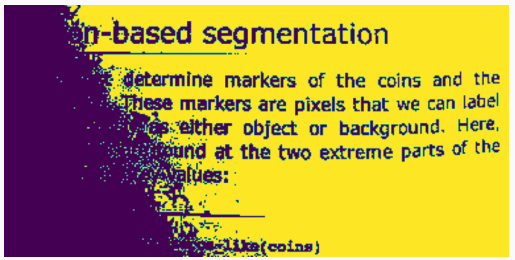
```
from skimage.filters import threshold_otsu, threshold_local

image = data.page()
global_thresh = threshold_otsu(image)
binary_global = image > global_thresh
block_size = 35
local_thresh = threshold_local(image, block_size, offset=10)
binary_local = image > local_thresh
fig, axes = plt.subplots(ncols=3, figsize=(7, 8))
ax = axes.ravel()
ax[0].imshow(image)
ax[0].set_title('Original')
ax[1].imshow(binary_global)
ax[1].set_title('Global thresholding')
ax[2].imshow(binary_local)
ax[2].set_title('Local thresholding')

for a in ax:
    a.set_axis_off()

cp.store_fig("local-threshold", close=True)
```

Histogram Operations

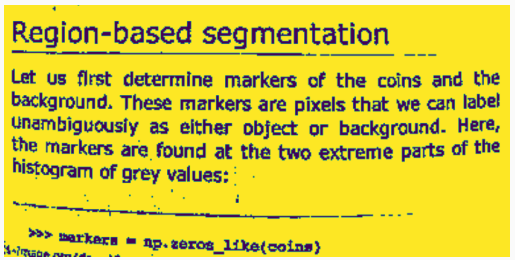
Original



Global thresholding



Local thresholding



Histogram Operations

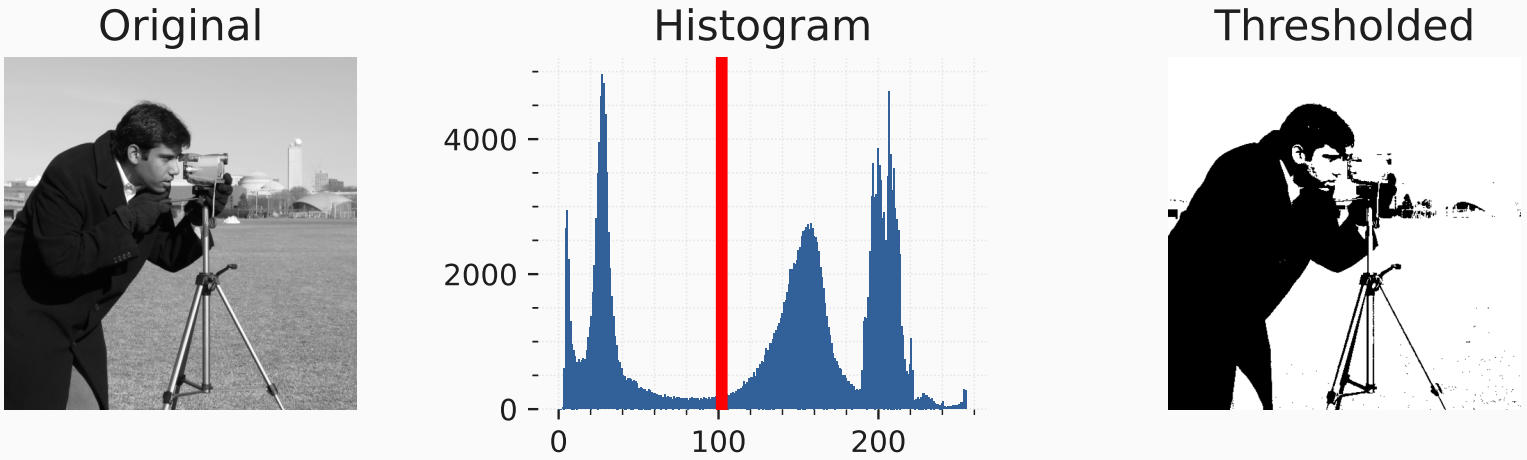


Figure 5

Figure 6: A comparison of the original image with the one with a mean threshold applied.

Histogram Operations

- This example converts an image with RGB channels into an image with a single grayscale channel.
- The value of each grayscale pixel is calculated as the weighted sum of the corresponding red, green and blue pixels as:

$$Y = 0.2125\ R + 0.7154\ G + 0.0721\ B$$

These weights are used by CRT phosphors as they better represent human perception of red, green and blue than equal weights.

Histogram Operations

```
import matplotlib.pyplot as plt

from skimage import data
from skimage.color import rgb2gray

original = data.astronaut()
grayscale = rgb2gray(original)

fig, axes = plt.subplots(1, 2, figsize=(8, 4))
ax = axes.ravel()

ax[0].imshow(original)
ax[0].set_title("Original")
ax[1].imshow(grayscale, cmap=plt.cm.gray)
ax[1].set_title("Grayscale")
cp.store_fig("rgb-to-grayscale", close=True)
```

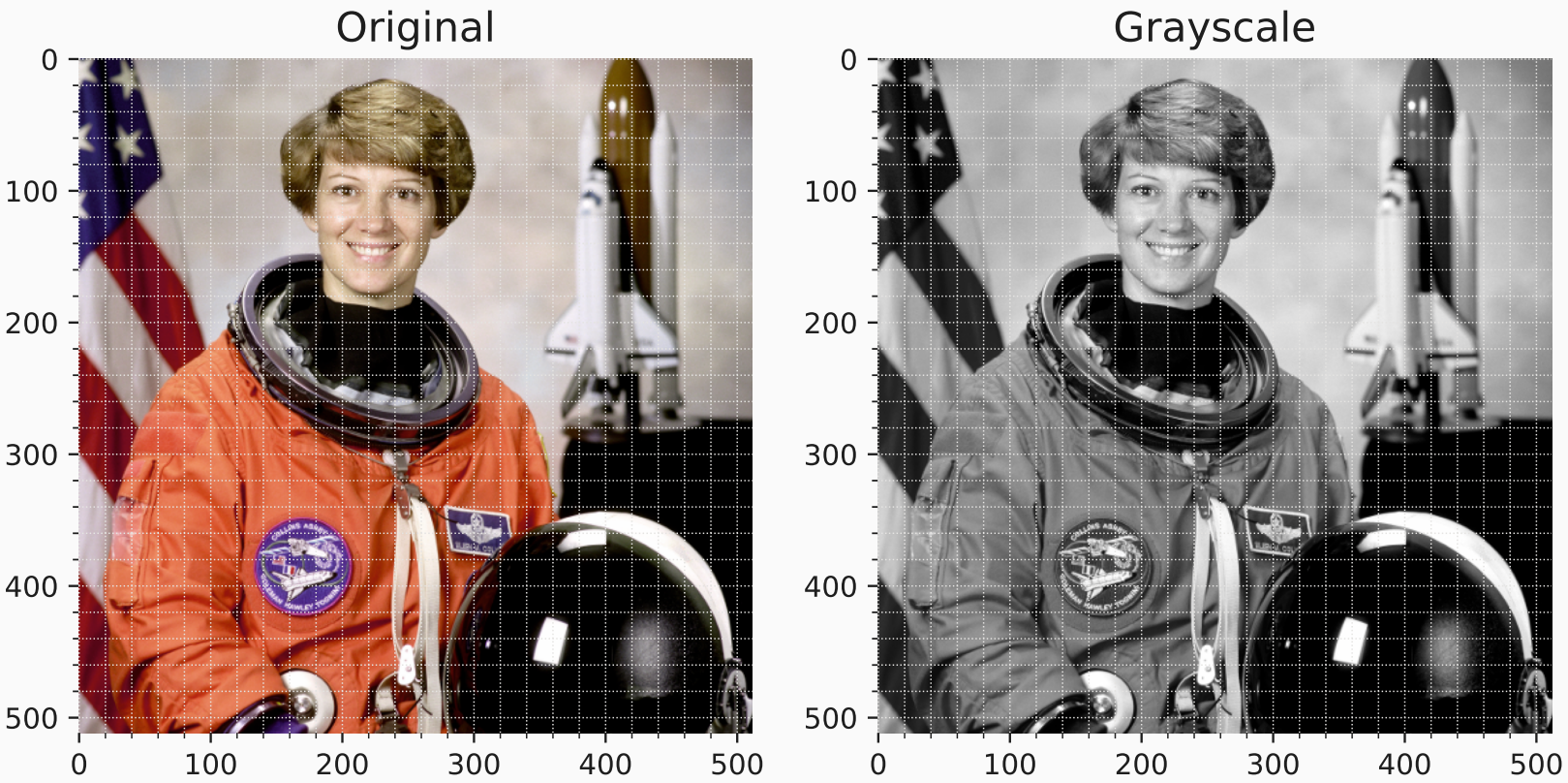


Figure 7: A comparison of an RGB image (left) v. its grey-scale version (right).

Histogram Operations

- Histogram matching manipulates the pixels of an input image so that its histogram matches the histogram of the reference image.
- If the images have multiple channels, the matching is done independently for each channel, as long as the number of channels is equal in the input image and the reference.
- Histogram matching can be used as a lightweight normalisation for image processing, such as feature matching, especially in circumstances where the images have been taken from different sources or in different conditions (i.e. lighting).

Histogram Operations

```
import matplotlib.pyplot as plt
from skimage import data, exposure
from skimage.exposure import match_histograms

reference,image  = data.coffee(), data.chelsea()
matched = match_histograms(image, reference, channel_axis=-1)

fig, (ax1, ax2, ax3) = plt.subplots(
    nrows=1, ncols=3, figsize=(8, 3), sharex=True, sharey=True
)
for aa in (ax1, ax2, ax3):
    aa.set_axis_off()

ax1.imshow(image); ax1.set_title('Source')
ax2.imshow(reference); ax2.set_title('Reference')
ax3.imshow(matched); ax3.set_title('Matched')

cp.store_fig("histogram-matching", close=True)
```

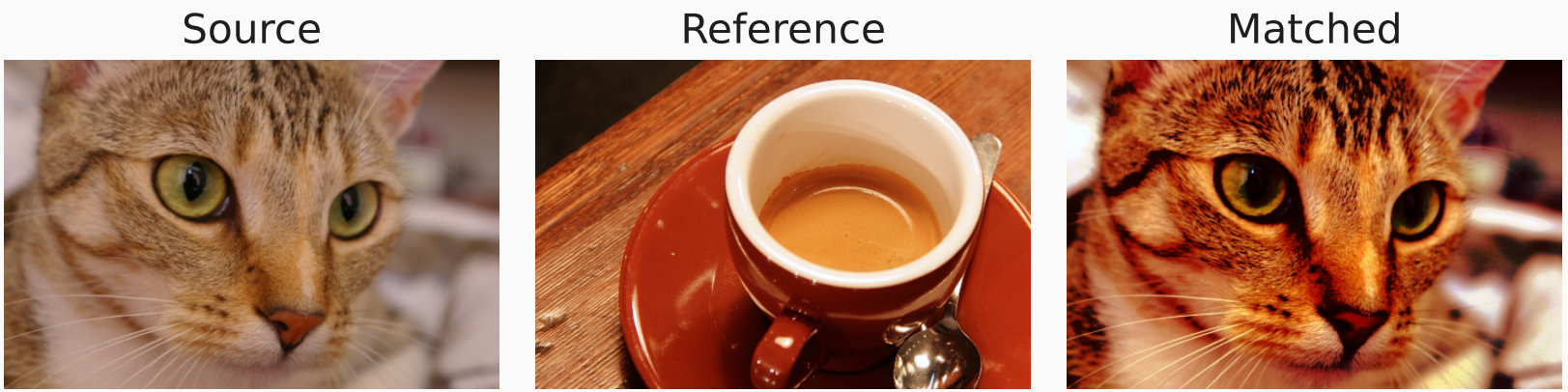


Figure 8

Histogram Operations

- Here, we identify and track the solid-liquid (S-L) interface in a nickel-based alloy undergoing solidification.
- Tracking the solidification over time enables the calculation of the solidification velocity.
- This is important to characterise the solidified structure of the sample and will be used to inform research into additive manufacturing of metals.

> The image sequence was obtained by the Center for Advanced Non-Ferrous Structural Alloys (CANFSA) using synchrotron x-radiography at the Advanced Photon Source (APS) at Argonne National Laboratory (ANL).

Histogram Operations

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import plotly.express as px
import plotly.io

from skimage import filters, measure, restoration
from skimage.data import nickel_solidification

image_sequence = nickel_solidification()
y0, y1, x0, x1 = 0, 180, 100, 330
image_sequence = image_sequence[:, y0:y1, x0:x1]

print(f'shape: {image_sequence.shape}')
```

```
shape: (11, 180, 230)
```

Compute Image Deltas

- The dataset is a 2D image stack with 11 frames (time points).
- We visualize and analyze it in a workflow where the first image processing steps are performed on the entire three-dimensional dataset (i.e., across space and time),
- the removal of localized, transient noise is favored as opposed to that of physical features (e.g., bubbles, splatters, etc.), which exist in roughly the same position from one frame to the next.

```
fig = px.imshow(
    image_sequence,
    animation_frame=0,
    binary_string=True,
    labels={'animation_frame': 'time point'},
)
plotly.io.show(fig)
```

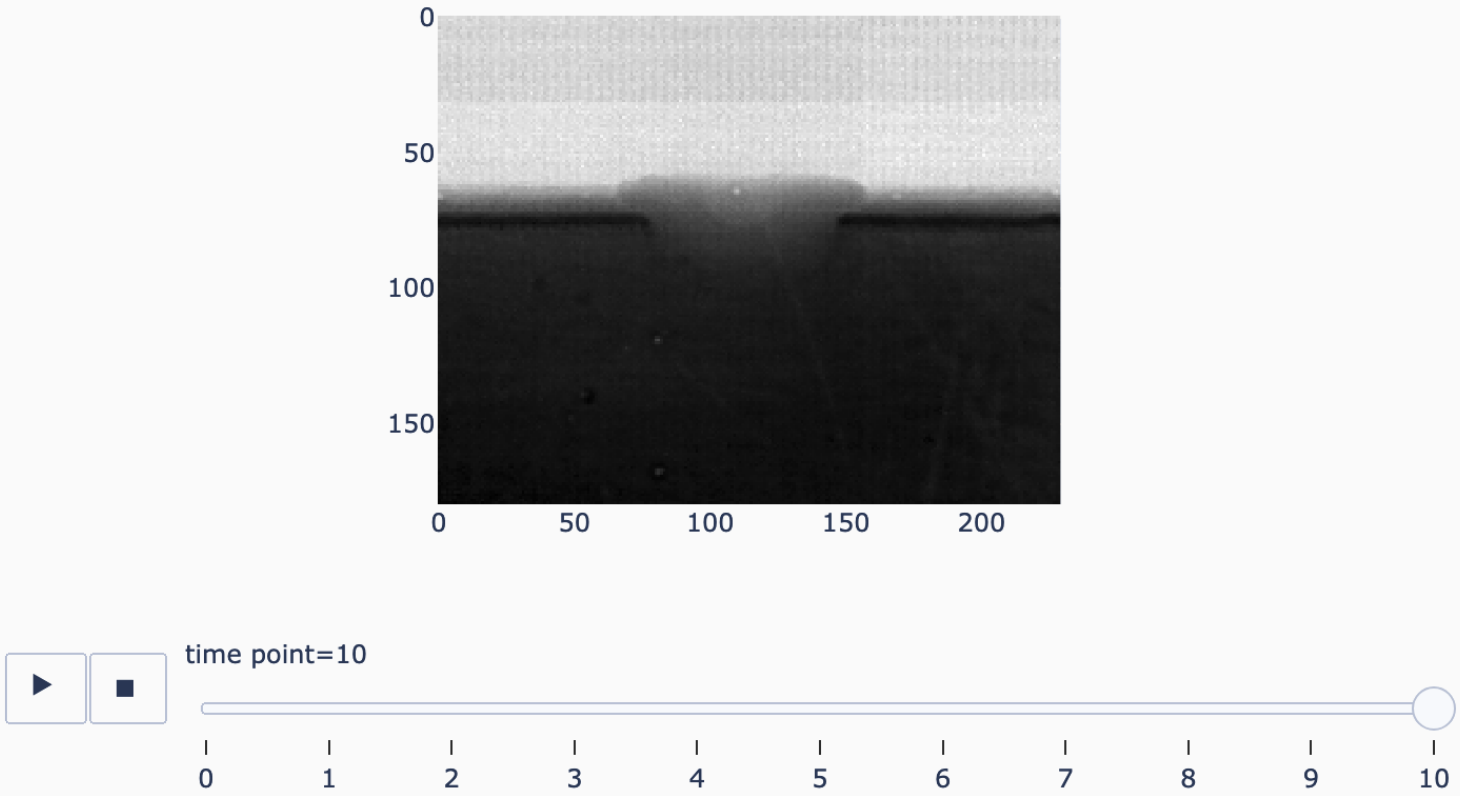


Figure 9

Compute image deltas

- First, apply a low-pass to smooth the images and reduce noise.
- Next, compute the image deltas.
- To do this, subtract the image sequence ending at the second-to-last frame from the image sequence starting at the second frame.

```
smoothed = filters.gaussian(image_sequence)
image_deltas = smoothed[1:, :, :] - smoothed[:-1, :, :]

fig = px.imshow(
    image_deltas,
    animation_frame=0,
    binary_string=True,
    labels={'animation_frame': 'time point'},
)
plotly.io.show(fig)
```

Histogram Operations

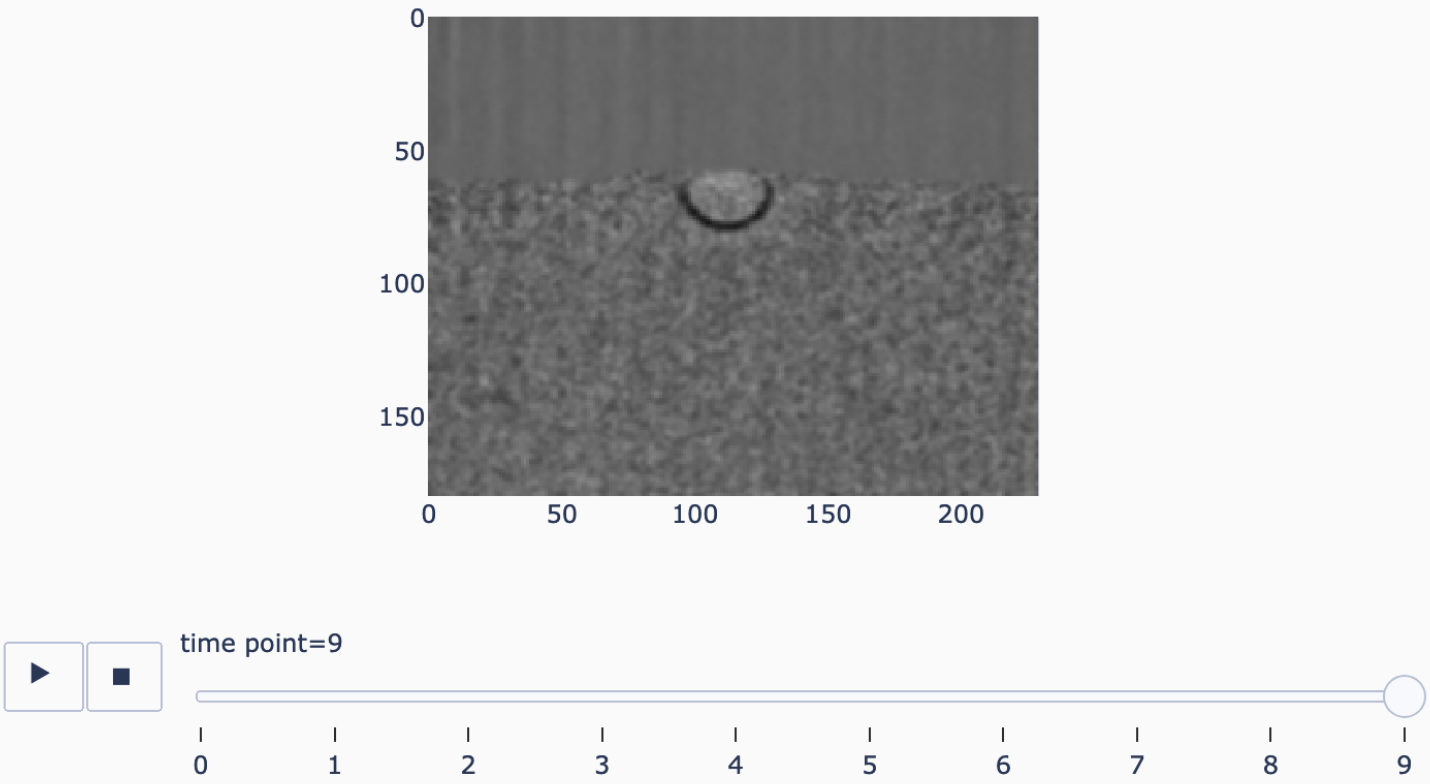


Figure 10

Clip Lowest and Highest intensities

- We now calculate the 5th and 95th percentile intensities of `image_deltas`,
- We want to clip the intensities which lie below the 5th percentile intensity and above the 95th percentile intensity, while also rescaling the intensity values to [0, 1].

Histogram Operations

```
p_low, p_high = np.percentile(image_deltas, [5, 95])
clipped = image_deltas - p_low
clipped[clipped < 0.0] = 0.0
clipped = clipped / p_high
clipped[clipped > 1.0] = 1.0

fig = px.imshow(
    clipped,
    animation_frame=0,
    binary_string=True,
    labels={'animation_frame': 'time point'},
)
plotly.io.show(fig)
```

Histogram Operations

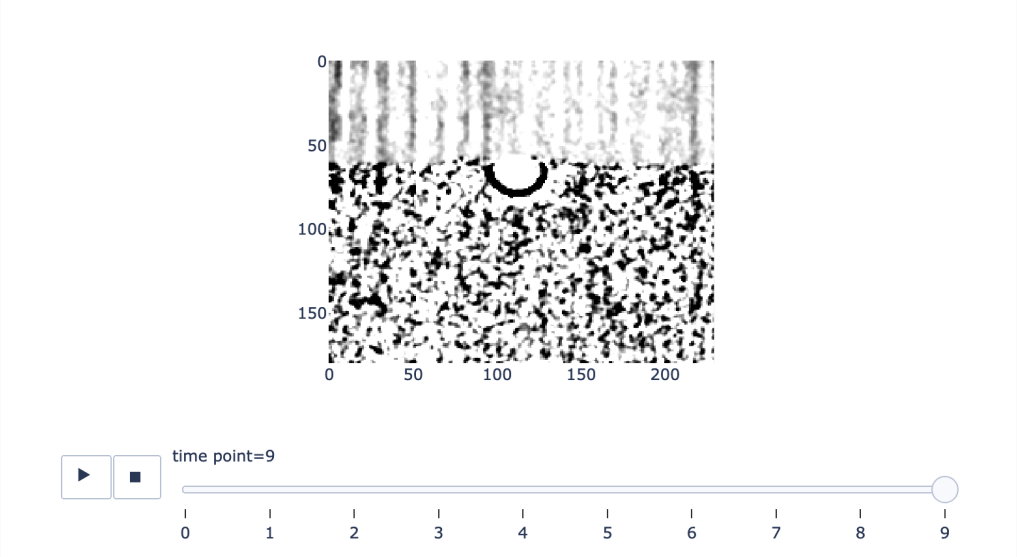


Figure 11

Invert and De-noise

- We invert the `clipped` images so the regions of highest intensity will include the region we are interested in tracking (i.e., the S-L interface). Then, we apply a total variation denoising filter to reduce noise beyond the interface.

```
inverted = 1 - clipped
denoised = restoration.denoise_tv_chambolle(inverted, weight=0.15)

fig = px.imshow(
    denoised,
    animation_frame=0,
    binary_string=True,
    labels={'animation_frame': 'time point'},
)
plotly.io.show(fig)
```

Histogram Operations

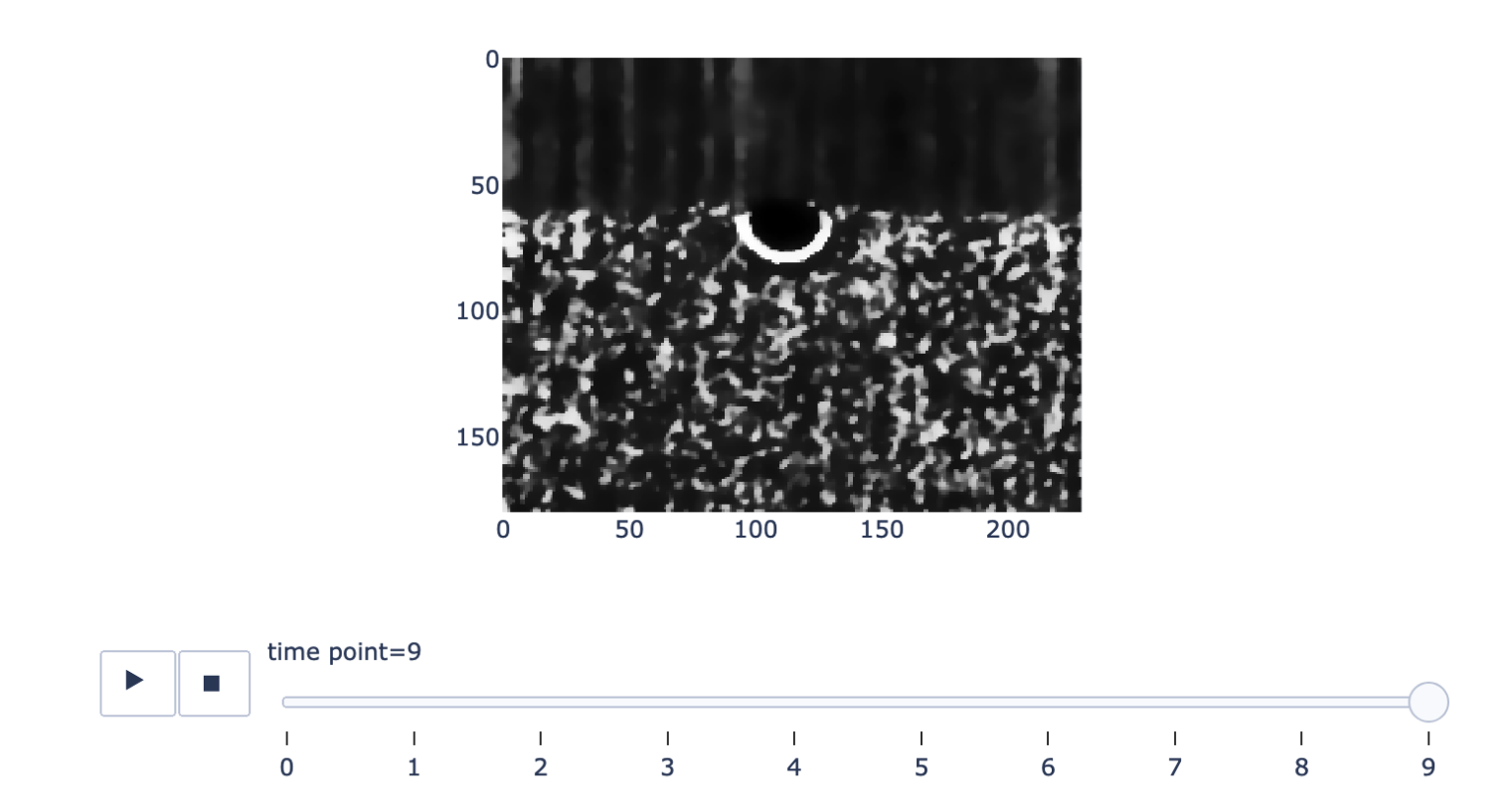


Figure 12

Binarise

- Our next step is to create binary images, splitting the images into foreground and background.
- We want the S-L interface to be the most prominent feature in the foreground of each binary image, so that it can eventually be separated from the rest of the image.
- We need a threshold value `thresh_val` to create our binary images, binarized.

> This can be set manually, but we shall use an automated minimum threshold method from the filters submodule of scikit-image.

Select the Largest Region

- The S-L interface appears as the largest region of connected pixels.
- For this step of the workflow, operate on each 2D image separately, as opposed to the entire 3D dataset.

> We are only interested in a single moment in time for each region.

Histogram Operations

- Apply `skimage.measure.label()` on the binary images so each region has its own label.
- Then, select the largest region in each image by computing region properties:
  - including the area property,
  - sorting by area values
- `skimage.measure.regionprops_table()` returns a table of region properties which can be read into a Pandas `DataFrame`.
- To begin with, consider the first image delta at this stage of the workflow, `binarized[0, :, :]`.

Histogram Operations

```
labeled_0 = measure.label(binarized[0, :, :])
props_0 = measure.regionprops_table(labeled_0,
                                    properties=('label', 'area',
                                    ↪  'bbox'))
props_0_df = pd.DataFrame(props_0)
props_0_df = props_0_df.sort_values('area', ascending=False)
# Show top five rows
print(props_0_df.head())
```

```
     label   area  bbox-0  bbox-1  bbox-2  bbox-3
1        2  417.0      60      83      91     144
198    199  235.0     141     141     179     165
11      12  136.0      62     164      87     180
9       10  122.0      61     208      79     229
8        9  114.0      61     183      83     198
```

Histogram Operations

- We can then select the largest region by matching its label with the one found in the first row of the sorted table.
- Let us visualize it, along with its bounding box (`bbox`) in red.

```
largest_region_0 = labeled_0 == props_0_df.iloc[0]['label']
minr, minc, maxr, maxc = (props_0_df.iloc[0][f'bbox-{i}'] for i in
↪  range(4))
fig = px.imshow(largest_region_0, binary_string=True)
fig.add_shape(type='rect', x0=minc, y0=minr, x1=maxc, y1=maxr,
↪  line=dict(color='Red'))
plotly.io.show(fig)
```

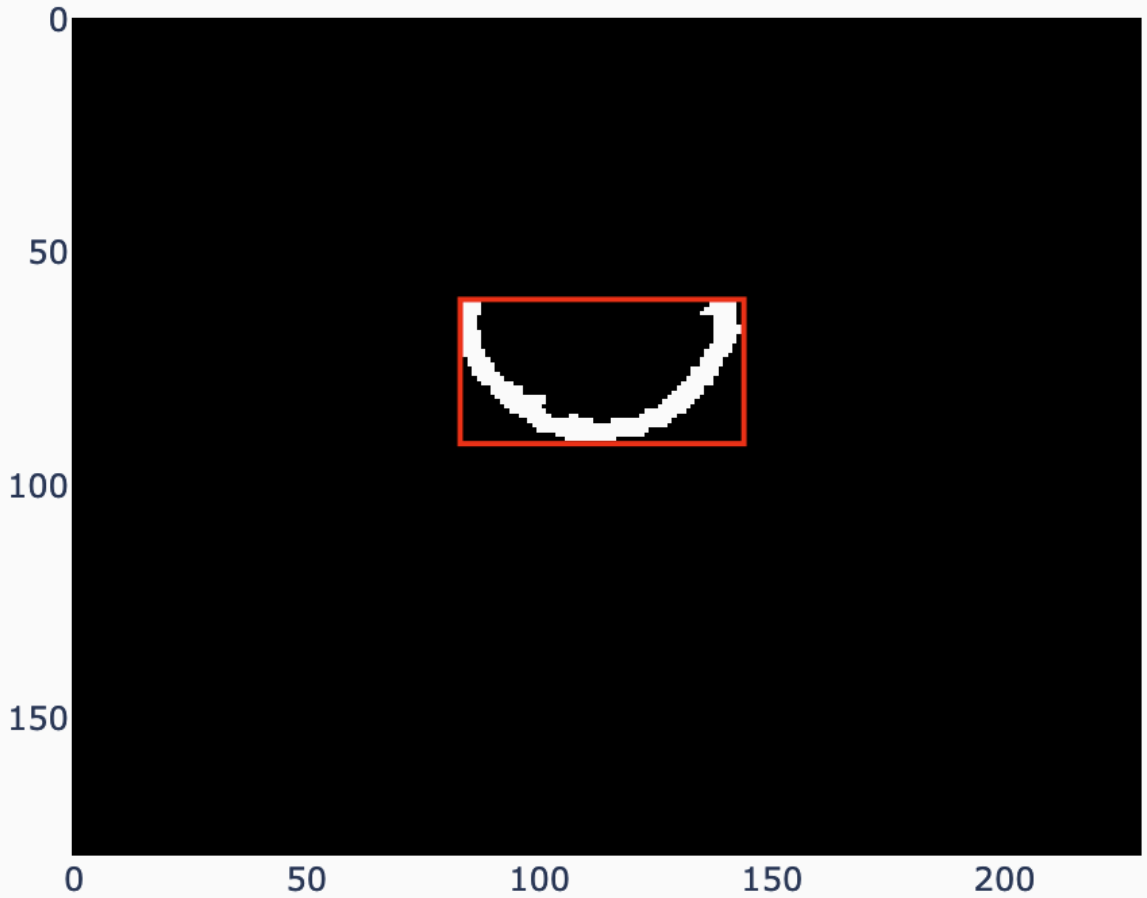


Figure 13: Showcasing the bounding box over the largest connected pixel group.

Histogram Operations

- We can see how the lower bounds of the box align with the bottom of the S-L interface by overlaying the same bounding box onto the 0th raw image.
- This bounding box was calculated from the image delta between the 0th and 1st images, but the bottom-most region of the box corresponds to the location of the interface earlier in time (0th image) because the interface is moving upward.

```
fig = px.imshow(image_sequence[0, :, :], binary_string=True)
fig.add_shape(type='rect', x0=minc, y0=minr, x1=maxc, y1=maxr,
↪  line=dict(color='Red'))
plotly.io.show(fig)
```

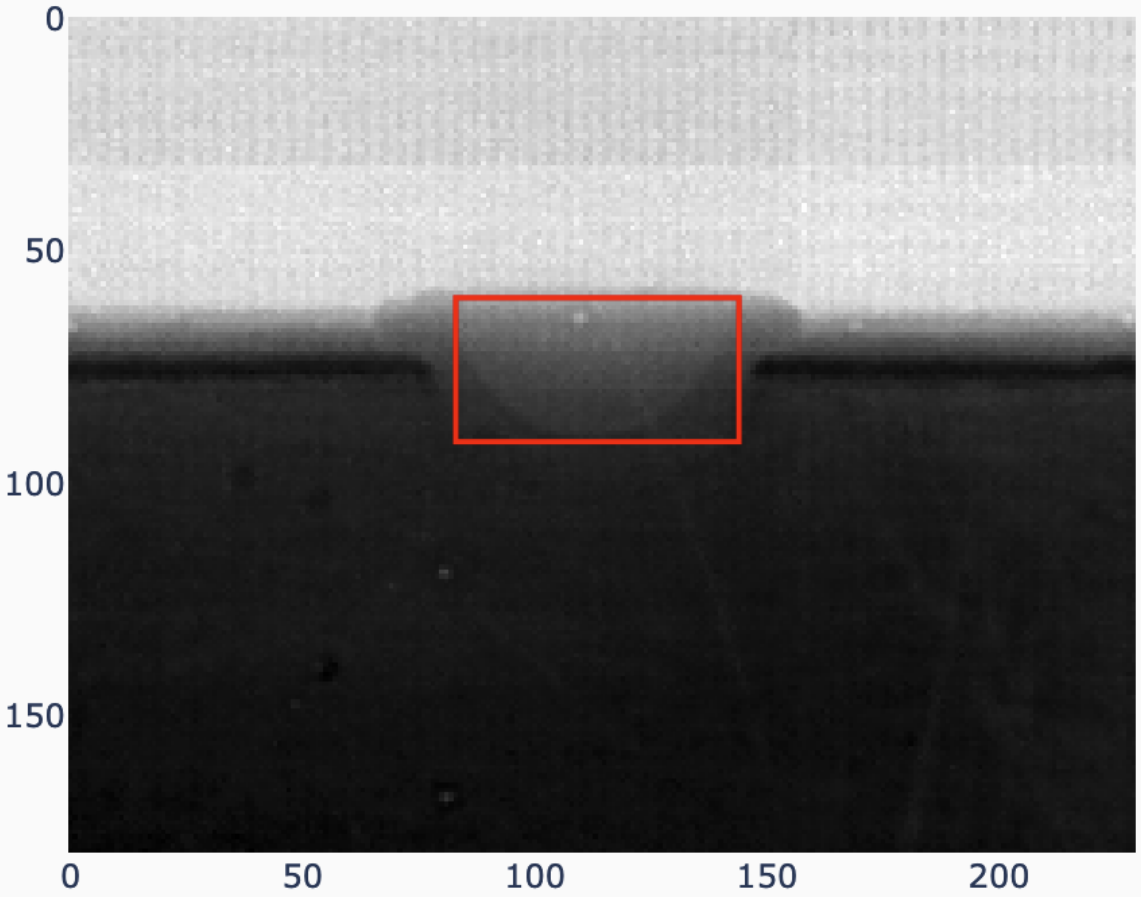


Figure 14: The bounding box applied to an original frame instead of a delta.

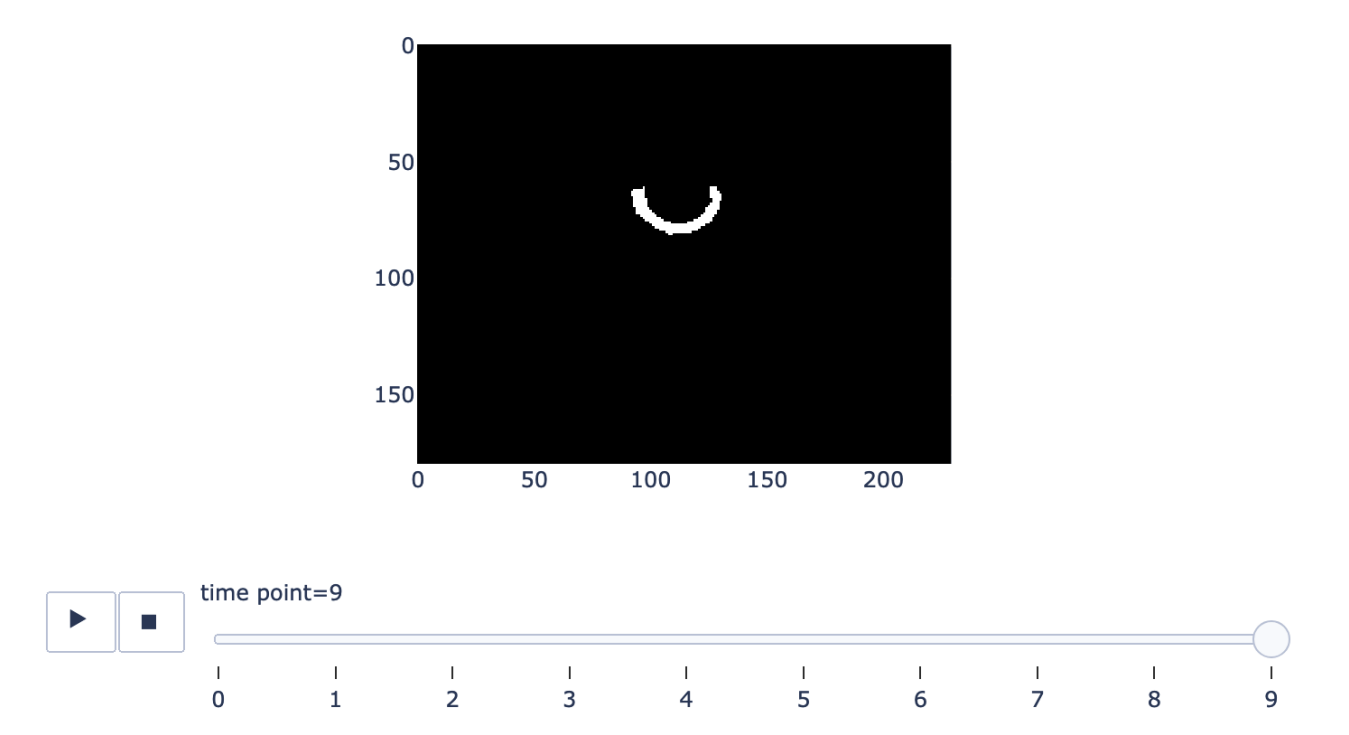


Figure 15

Plot S-L v. time

- The final step is to plot the location of the **S-L interfaces** over time.
- This can be achieved by plotting `maxr` (third element in `bbox`) over time as this value shows the y location of the bottom of the interface.
- The pixel size in this experiment was 1.93 microns and the frame-rate was 80,000 frames per second, so these values are used to convert pixels and image number to physical units.
- We calculate the average solidification velocity by fitting a linear polynomial to the scatter plot.
- The velocity is the first-order coefficient.

Histogram Operations

```
largest_region = np.empty_like(binarized)
bboxes = []

for i in range(binarized.shape[0]):
    labeled = measure.label(binarized[i, :, :])
    props = measure.regionprops_table(labeled, properties=('label',
    ↪  'area', 'bbox'))
    props_df = pd.DataFrame(props)
    props_df = props_df.sort_values('area', ascending=False)
    largest_region[i, :, :] = labeled == props_df.iloc[0]['label']
    bboxes.append([props_df.iloc[0][f'bbox-{i}'] for i in
    ↪  range(4)])
fig = px.imshow(
    largest_region,
    animation_frame=0,
    binary_string=True,
    labels={'animation_frame': 'time point'},
)
plotly.io.show(fig)
```

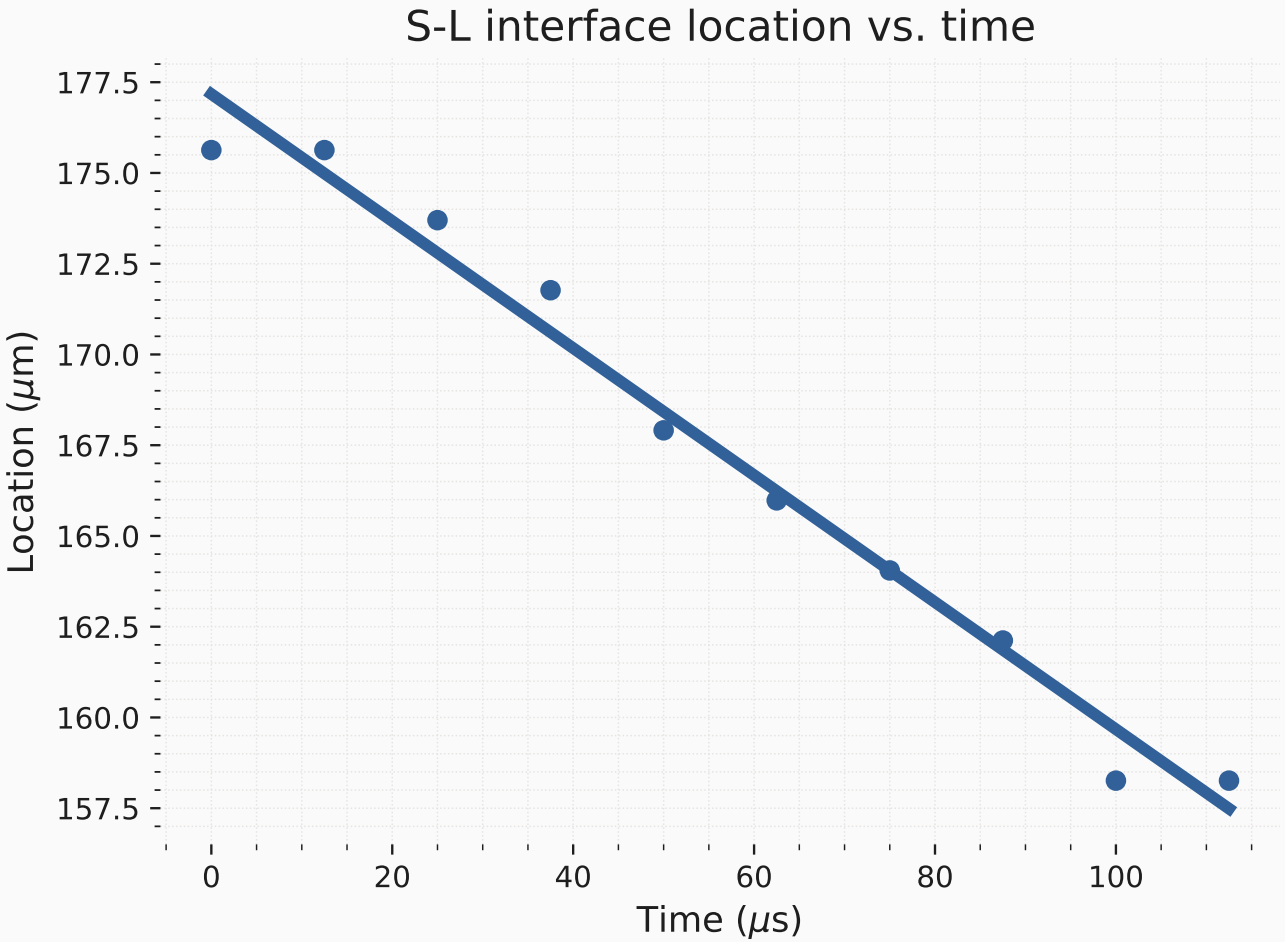


Figure 16: Using the image alone, it was possible to determine the metals S-L state transition.

Steganography is the practice of concealing a message within another message or a physical object.

- Today, we are going to do an implementation.
- We will make a program which writes text to images and reads it.
- We assign pixel values to each possible character.
- This is a proof of concept but anyone is welcome to improve on it.

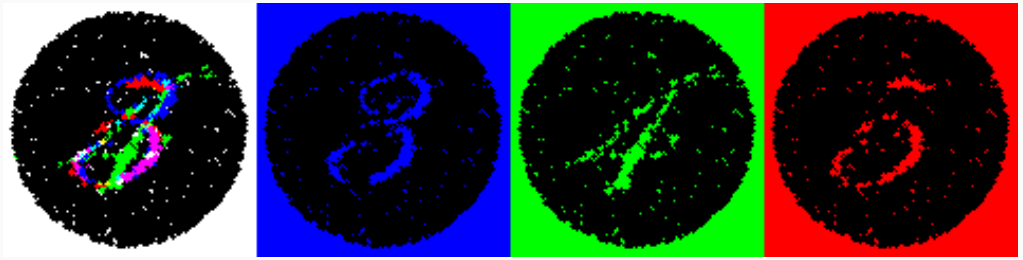


Figure 17: The same image viewed by white, blue, green, and red lights reveals different hidden numbers.

Histogram Operations

```
from PIL import Image # For reading images
import string # constants containing letters and digits.
```

```
# Open the image.
image = Image.open("Fruit.jpg")

# Get the image size and save it.
width, height = image.size
startingPixel = (10, 10, 255)
endingPixel = (255, 20, 20)
# Dictionary which hold the relations between pixels and characters
lettersToPixels = {}
pixelsToLetters = {}
# Making the relations using the string library
for i, lttr in enumerate(string.ascii_letters + string.digits + '
↪   '):
    lettersToPixels[lttr] = i
    pixelsToLetters[i] = lttr
```

Histogram Operations

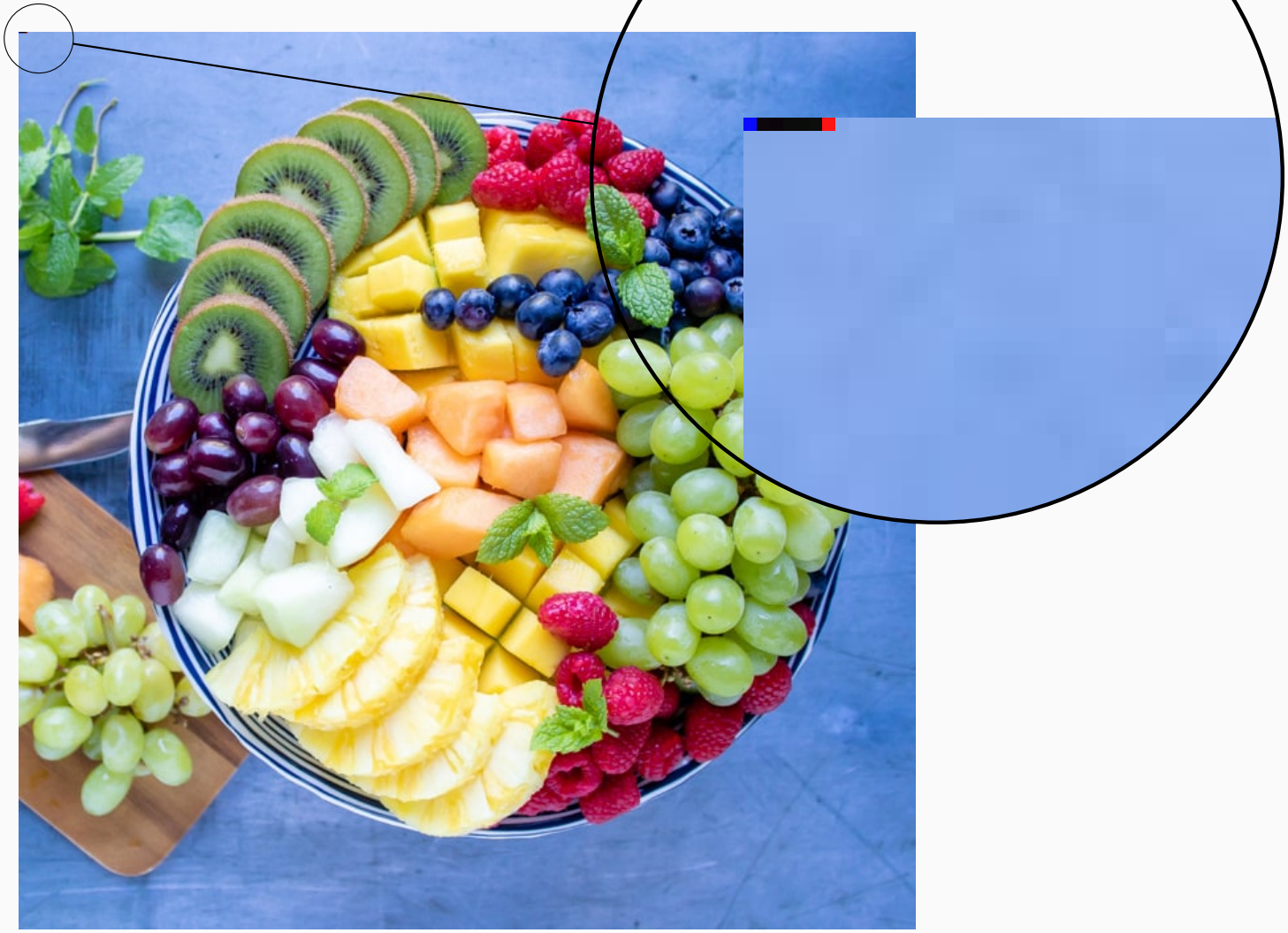
- Let's get to hiding messages in images.
  - If the mode specified by the user is write, user wants to write.
- But we also check if the text was set.
- After that, assert if the text length is greater than the image width.
- Then, draw the starting pixel onto the image at position (0, 0), which is the top left.
- Loop until the message is written.

Histogram Operations

```
def write(message):
    assert len(message) < width
    # Draw the starting Pixel
    image.putpixel((0, 0), startingPixel)

    # Draw a pixel for each letter in the test
    for index, letter in enumerate(message):

        # The Middle value (g = green) is the number in the
        ↪  dictionary
        image.putpixel((index+1, 0), (11, lettersToPixels[letter],
        ↪  11))

            # Draw the ending Pixel
    image.putpixel((index+2, 0), endingPixel)

        # Save the image to the same place
    image.save("images/Histogram-Operations/encrypted-fruit.png")

write("hello")
```

Figure 18: Encrypted image. Can you spot where the information is hidden?

Histogram Operations



Histogram Operations

```
def read():
    image =
    ↪  Image.open("images/Histogram-Operations/encrypted-fruit.png")
    text = ''
    index = 1
    while True:
        # Loop throug each pixel in the top row
        pixel = image.getpixel((index, 0))
        # if the pixel is the ending pixel, we break the loop
        if pixel == endingPixel:
            break
        try:
            # Get the letter from the dictionary with the g value.
            text += pixelsToLetters[pixel[1]]
        except:
            pass
        index += 1
         # Print out the text
    print(text)
read()
```

Histogram Operations

```
hello
```

Appendix

Go Back

- Univariate object is an expression, equation, function or polynomial involving only **one variable**.
- Objects involving more than one variable are multivariate.
  - In statistics, a univariate distribution characterizes one variable, although it can be applied in other ways as well.
    - i.e., in time series analysis, the whole time series is the "variable": a univariate time series is the series of values over time of a single quantity.

Go Back

In probability theory, the sample space (also called sample description space,[1] possibility space,[2] or outcome space[3]) of an experiment or random trial is the set of all possible outcomes or results of that experiment.[4] A sample space is usually denoted using set notation, and the possible ordered outcomes, or sample points,[5] are listed as elements in the set. It is common to refer to a sample space by the labels S, Ω, or U (for "universal set"). The elements of a sample space may be numbers, words, letters, or symbols. They can also be finite, countably infinite, or uncountably infinite.[6]

Go Back

Stands for red green blue alpha.

While it is sometimes described as a color space, it is actually a three-channel RGB color model supplemented with a fourth alpha channel.

Alpha indicates how opaque each pixel is and allows an image to be combined over others using alpha compositing, with transparent areas and anti-aliasing of the edges of opaque regions. Each pixel is a 4D vector.

Go Back

There are a wide variety of data types used in `opencv`. The most common are:

Go Back

Aperture of an optical system is a hole or an opening that primarily limits light propagated through the system.

Figure 19: Different apertures of a lens.

Go Back

The Gaussian function $g(x)$, is defined as:

$$f(x) = \exp\left(-ax^2\right)$$

The Fourier transform $(\mathcal{F}_x)$ if given by

$$\begin{aligned}
\mathcal{F}_x(k) &= \int_{-\infty}^{\infty} \exp\left(-ax^2\right) \exp\left(-2\pi \mathbf{j}\, kx\right)\, dx, \\
&= \int_{-\infty}^{\infty} \exp\left(-ax^2\right) \left[\cos 2\pi kx - \mathbf{j} \sin 2\pi kx\right]\, dx \\
&= \int_{-\infty}^{+\infty} \exp\left(-ax^2\right) \cos 2\pi kx\, dx - \mathbf{j} \int_{-\infty}^{+\infty} \exp\left(-ax^2\right) \sin 2\pi kx\, dx.
\end{aligned}$$

Go Back

The second integrand is odd, so integration over a symmetrical range gives 0.

The value of the first integral is given by Abramowitz and Stegun, so:

$$\mathcal{F}_x(k) = \sqrt{\frac{\pi}{a}} \exp\left(\frac{-\pi^2 k^2}{a}\right),$$

therefore a Gaussian transforms to another Gaussian ■

Go Back

The Weierstrass transform of a function $f : \mathbf{R} \to \mathbf{R}$, is a smoothed version of f(x) obtained by averaging the values of f, weighted with a Gaussian centred at x.

The function F is defined by:

$$F(x) = \frac{1}{\sqrt{4\pi}} \int_{-\infty}^{\infty} f(y)\, e^{-\frac{(x-y)^2}{4}}\, dy = \frac{1}{\sqrt{4\pi}} \int_{-\infty}^{\infty} f(x-y)\, e^{-\frac{y^2}{4}}\, dy$$

the convolution of f with the Gaussian function

$$\frac{1}{\sqrt{4\pi}} e^{-x^2/4}$$

The factor $\frac{1}{\sqrt{4\pi}}$ is chosen so that the Gaussian will have a total integral of 1, with the consequence that constant functions are not changed by the Weierstrass transform.

Go Back

Bivariate analysis is a form of quantitative analysis which involves the analysis of two variables, for the purpose of determining the empirical relationship between them.

Bivariate analysis can be helpful in testing simple hypotheses of association. Bivariate analysis can help determine to what extent it becomes easier to know and predict a value for one variable (possibly a dependent variable) if we know the value of the other variable (possibly the independent variable) (see also correlation and simple linear regression).

Go Back

The central limit theorem (CLT) states that, under appropriate conditions, the distribution of a normalized version of the sample mean converges to a standard normal distribution. This holds even if the original variables themselves are not normally distributed. There are several versions of the CLT, each applying in the context of different conditions.

The theorem is a key concept in probability theory because it implies that probabilistic and statistical methods that work for normal distributions can be applicable to many problems involving other types of distributions.

Go Back

A varifocal lens is a camera lens with variable focal length in which focus changes as focal length (and magnification) changes, as compared to a parfocal ("true") zoom lens, which remains in focus as the lens zooms (focal length and magnification change).

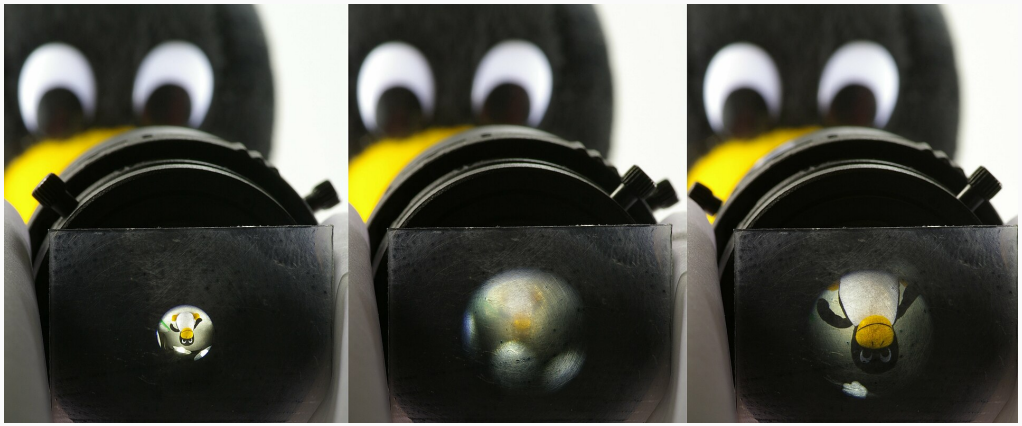


Figure 20: A varifocal lens. Left image is at 2.8 mm, in focus. Middle image is at 12 mm with the focus left alone from 2.8 mm. Right image is at 12 mm refocused. The close knob is focal length and the far knob is focus.

Go Back

In photography and optics, vignetting (/v n j t ŋ/; vin-YET-ing) is a reduction of an image's brightness or saturation toward the periphery compared to the image center. The word vignette, from the same root as vine, originally referred to a decorative border in a book. Later, the word came to be used for a photographic portrait that is clear at the center and fades off toward the edges. A similar effect is visible in photographs of projected images or videos off a projection screen, resulting in a so-called "hotspot" effect.

Go Back

Impossible colors are colors that do not appear in ordinary vision.

Different color theories suggest different hypothetical colors that humans are incapable of perceiving for one reason or another, and fictional colors are routinely created in popular culture.

While some such colors have no basis in reality, phenomena such as cone cell fatigue enable colours to be perceived in certain circumstances that would not be otherwise.

Go Back

Generation loss is the loss of quality between subsequent copies or transcodes of data.

Anything that reduces the quality of the representation when copying, and would cause further reduction in quality on making a copy of the copy, can be considered a form of generation loss.

File size increases are a common result of generation loss, as the introduction of artifacts may actually increase the entropy of the data through each generation.

Go Back

Shutter speed or exposure time is the length of time that the film or digital sensor inside the camera is exposed to light (that is, when the camera's shutter is open) when taking a photograph [**ray2002applied**]. The amount of light that reaches the film or image sensor is proportional to the exposure time. 1/500 of a second will let half as much light in as 1/250.

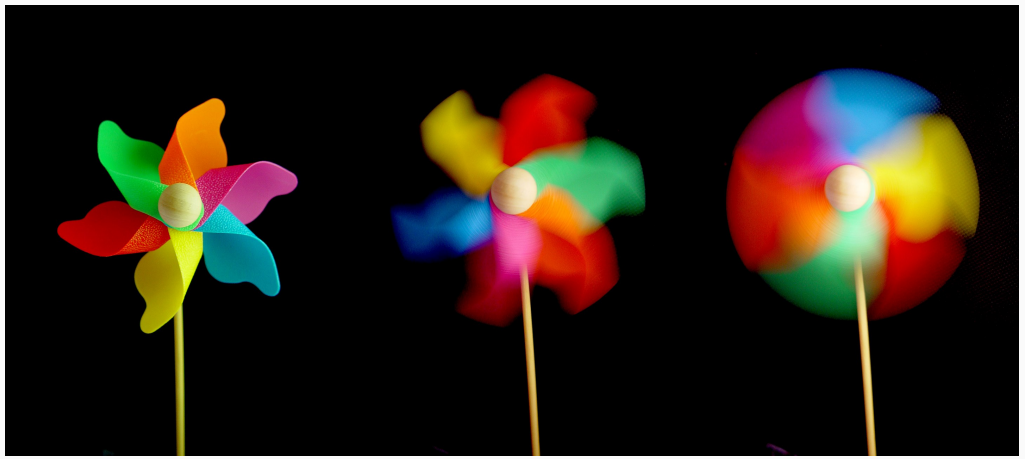


Figure 21: Effects of shutter speed on image [**Dilmen2004**]. As shutter speed slows down the image gets blurrier.

Go Back

Baron Siméon Denis Poisson **FRS FRSE** was a French mathematician and physicist who worked on statistics, complex analysis, partial differential equations, the calculus of variations, analytical mechanics, electricity and magnetism, thermodynamics, elasticity, and fluid mechanics.

Moreover, he predicted the Arago spot in his attempt to disprove the wave theory of Augustin-Jean Fresnel.

21 June 1781 – 25 April 1840

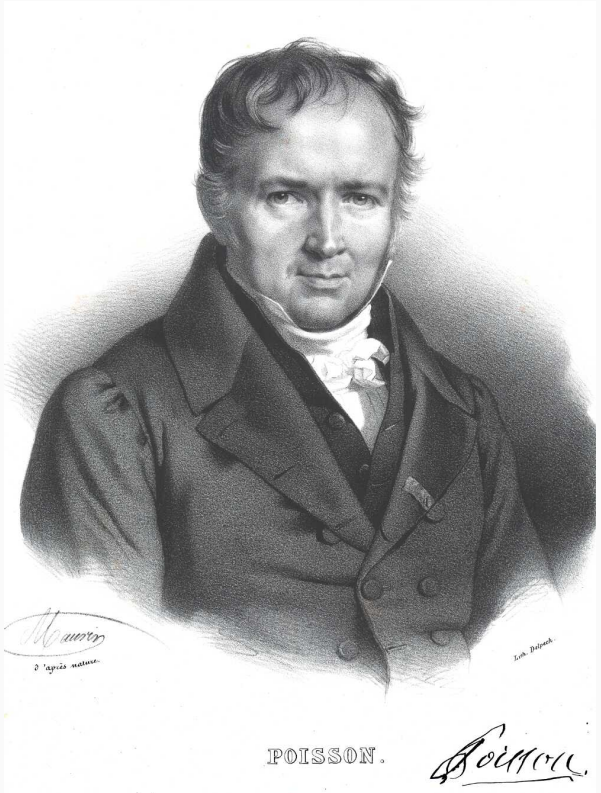


Figure 22: Baron Siméon Denis Poisson.

Go Back

In optics, the Arago spot, Poisson spot, or Fresnel spot is a bright point that appears at the center of a circular object's shadow due to Fresnel diffraction.

This spot played an important role in the discovery of the wave nature of light and is a common way to demonstrate that light behaves as a wave.

Go Back

An anonymous function in Python is a function without a name. It can be immediately invoked or stored in a variable. Anonymous functions in Python are also known as lambda functions.

Appendix

Slides were created using **GNU Emacs** version 29.1 with **AUCTeX** 14.0.7.

"Emacs, is a family of text editors that are characterised by their extensibility. The manual for the most widely used variant, GNU Emacs, describes it as "the extensible, customizable, self-documenting, real-time display editor."

"AUCTeX is a package for writing and formatting TeX files in GNU Emacs."

Beamer class was used as template with the **LuaTeX** engine.

"Beamer is a LaTeX class for generating slides."

"LuaTeX is a TeX-based computer typesetting system which started as a version of pdfTeX with a Lua scripting engine embedded."

All code presented in lectures are in **Python**, using version 3.9.13.

"Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation."

Appendix i