Quantitative Big Imaging - Dynamic experiments

Anders Kaestner

May 11, 2023

CONTENTS

This is the lecture notes fot the 9th lecture of the Quantitative big imaging class given during the spring semester 2021 at ETH Zurich, Switzerland.

0.1 Dynamic Experiments

```
import matplotlib.pyplot as plt
import seaborn as sns
from   skimage.morphology import label
from   skimage.measure import regionprops
import pandas as pd

from skimage.feature import corner_peaks, corner_harris, BRIEF
from skimage.transform import warp, AffineTransform
from skimage import data
from skimage.io import imread
from scipy.ndimage import distance_transform_edt
from skimage.filters import threshold_otsu

import numpy as np
import pydot

%load_ext autoreload
%autoreload 2
# plt.rcParams["figure.figsize"] = (8, 8)
# plt.rcParams["figure.dpi"] = 150
# plt.rcParams["font.size"] = 14
plt.rcParams['font.family'] = ['sans-serif']
plt.rcParams['font.sans-serif'] = ['DejaVu Sans']
plt.style.use('default')
sns.set_style("whitegrid", {'axes.grid': False})

%config InlineBackend.figure_format = 'retina'
```

0.1.1 Literature / Useful References

Books

- John C. Russ, “The Image Processing Handbook”,(Boca Raton, CRC Press)
- A. Ardeshir Goshtasby, “Image Registration Principles, Tools and Methods (Springer Verlag)
- B. Jähne,”Spatio-Temporal Image Processing” ,(Springer Verlag)

Papers / Sites

- Comparsion of Tracking Methods in Biology
- Chenouard, N., Smal, I., de Chaumont, F., Maška, M., Sbalzarini, I. F., Gong, Y., … Meijering, E. (2014). Objective comparison of particle tracking methods. Nature Methods, 11(3), 281–289. doi:10.1038/nmeth.2808
- Maska, M., Ulman, V., Svoboda, D., Matula, P., Matula, P., Ederra, C., … Ortiz-de-Solorzano, C. (2014). A benchmark for comparison of cell tracking algorithms. Bioinformatics (Oxford, England), btu080–. doi:10.1093/bioinformatics/btu080
- Keypoint and Corner Detection
- Distinctive Image Features from Scale-Invariant Keypoints - https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf
- https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html
- Registration
- https://itk.org/ITKSoftwareGuide/html/Book2/ITKSoftwareGuide-Book2ch3.html
- Multiple Hypothesis Testing
- Coraluppi, S. & Carthel, C. Multi-stage multiple-hypothesis tracking. J. Adv. Inf. Fusion 6, 57–67 (2011).
- Chenouard, N., Bloch, I. & Olivo-Marin, J.-C. Multiple hypothesis tracking in microscopy images. in Proc. IEEE Int. Symp. Biomed. Imaging 1346–1349 (IEEE, 2009).
- Keypoint and Corner Detection
- Distinctive Image Features from Scale-Invariant Keypoints - https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf
- https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html

0.1.2 Previously on QBI …

- Image Enhancment
- Highlighting the contrast of interest in images
- Minimizing Noise
- Understanding image histograms
- Automatic Methods
- Component Labeling
- Single Shape Analysis
- Complicated Shapes
- Distribution Analysis

0.1.3 Quantitative "Big" Imaging

The course has covered imaging enough and there have been a few quantitative metrics, but "big" has not really entered.

What does **big** mean?

- Not just / even large
- Being ready for *big data*
- The three V's
  - volume,
  - velocity,
  - variety
- scalable, fast, easy to customize

0.1.4 Outline

- Motivation (Why and How?)
- Scientific Goals

Experiments

- Simulations
- Experiment Design

0.2 Imaging of dynamic experiments

0.2.1 Dynamic processes

- Relating to objects in motion
- Characterized by continuous change, activity, or progress

0.2.2 Real time

- The actual time during which a process or event occurs.
- (Computing) of or relating to a system in which
  - input data is processed within milliseconds
  - so that it is available virtually immediately as feedback.

0.2.3 Imaging

Imaging is the

- capture,
- storage,
- manipulation,
- and display

of images.

0.2.4 Experiments

1. What sort of dynamic experiments do we have?
2. How can we design good dynamic experiments?

Fig. 1: Contiuous processes evolve over time without guarantee that it returns to the same point.

Fig. 2: Repetetive processes always behaves the same with a given frequency.

0.2.5 What information are you looking for?

We can say that it looks like, but many pieces of quantitative information are difficult to extract

- How fast is it going?
- How many particles are present?
- Are their sizes constant?
- Are some moving faster?
- Are they rearranging?

0.2.6 Image analysis

How can we…

1. track objects between points?
2. track shape?
3. track distribution?
4. track topology?
5. track voxels?
6. assess deformation and strain?
7. assess more general cases?

→ **How does this help answering your questions?**

0.3 Motivation spatiotemporal analysis

- 3D images are already difficult to interpret on their own
  - Obscured structures
  - Screens only show 2D
- 3D movies (4D) are almost impossible

0.3.1 2D movies can also be challenging

They are 3D!

- x, y, t

Example: a water jet

Example: coffee making

0.3.2 Other image series - 3D Reconstruction

Tomography

One of the most commonly used scans in the hospital is called a computed tomography scan.

This scan works by creating 2D X-ray projections in a number of different directions in order to determine what the 3D volume looks like

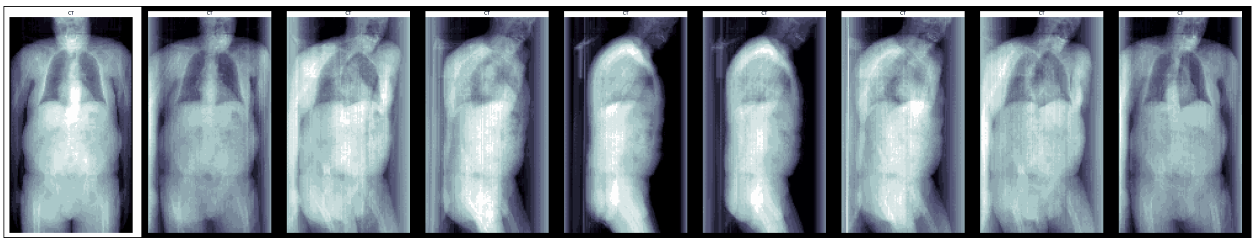


Fig. 1: Some views from a CT scan of a patient

Stage Tilting

Beyond just tracking we can take multiple frames of a still image and instead of looking for changes in the object, we can change the angle.

The pollen image below shows this for SEM

0.3.3 Scientific Goals of dynamic experiments

Rheology

Understanding the flow of liquids and mixtures is important for many processes

- blood movement in arteries, veins, and capillaries
- oil movement through porous rock
- air through dough when baking bread
- magma and gas in a volcano

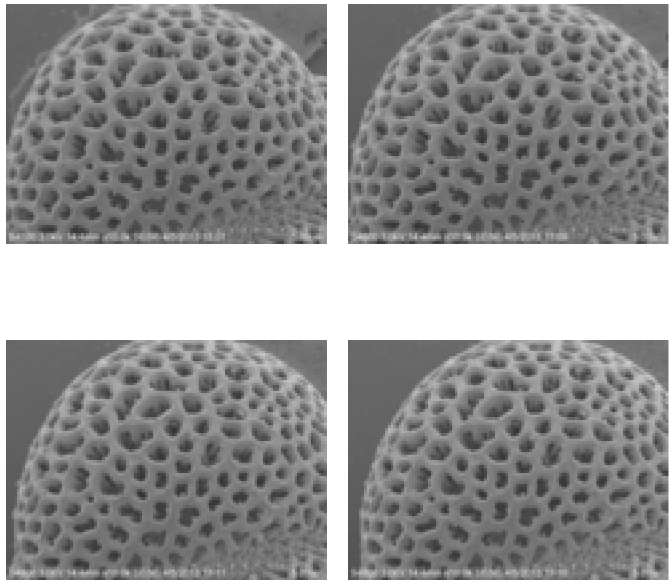


Fig. 2: A tilt series of a pollen grain.

Deformation

Deformation is similarly important since it plays a significant role in the following scenarios

- red blood cell lysis in artificial heart valves
- microfractures growing into stress fractures in bone
- toughening in certain wood types

0.4 Dynamic experiments

The first step of any of these analyses is proper experimental design. Since there is always:

- A limited field of view
- A voxel size
- A maximum rate of measurements
- Dose limitations
  - Sample damage
  - Limited flux
- A non-zero cost for each measurement

There are always trade-offs to be made between

- getting the best possible high-resolution nanoscale dynamics
- and capturing the system level behavior.

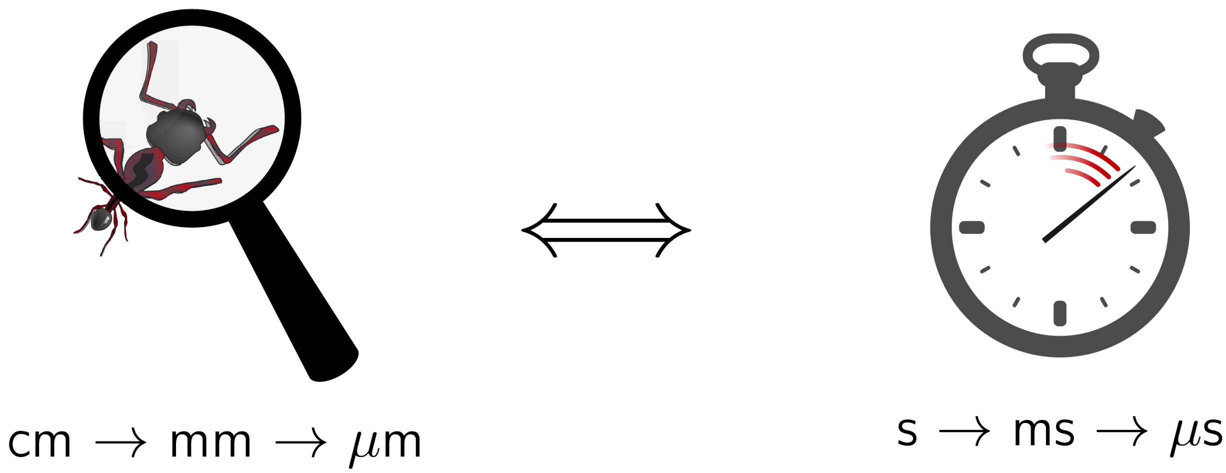


Fig. 1: The trade-off between time and resoultion.

0.4.1 Factors affecting the image quality

- Process speed
- Spatial resolution
- Intensity dynamics

Fig. 2: Parts moving too fast will introduce motion blur.

Example

You have a dose limited experiment

- 100μm pixels
- 1s Exposure time
- 800 neutrons per pixel

To maintain the SNR you have to

	Smaller pixels	Higher frame-rate
Modification	Longer exposure time	Larger pixels

Rebinning

Rebinning or using larger pixels can be used to improve the SNR.

…but you may miss small details

Fig. 3: Rebinning improve SNR at the cost of small features.

Frame rates

The sample rate needed to study a dynamic process must be chosen to allow several samples during the rise typically 5-20 times the fastest expected time constants in the system.

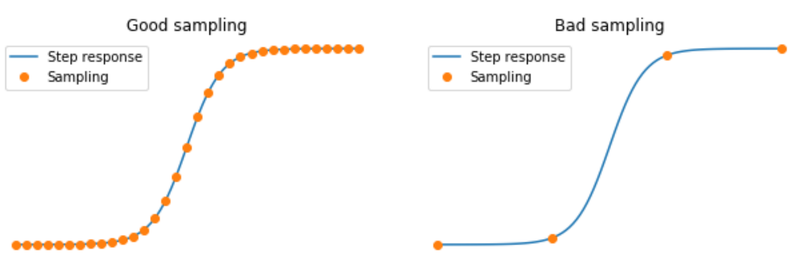


Fig. 4: Selecting the correct frame rate is essential to allow the study of a process.

Some processes can be controlled and even stopped at different levels. This allows measurements of steady intermediate steady state conditions.

Fig. 5: Some processes can be followed by piecewise constant measurements

0.4.2 Planning dynamic experiments

If we measure too fast

- sample damage
- miss out on long term changes
- have noisy data

Too slow

- miss small, rapid changes
- blurring and other motion artifacts

Too high resolution

- too few unique structures in field of view to track

Too low resolution

- not sensitive to small changes

Simple processes

For relatively simple processes you can extract images along the time axes. This will make the analysis easier.

Example Capillary rise

We want to study capillary rise and Washbourne's equation

Analysis steps

- Extract y-t slices
- Segment the front
- Locate the front as function of time
- Plot as function of $\sqrt{t}$

For the interested: Neutron imaging tutorial - capillary rise

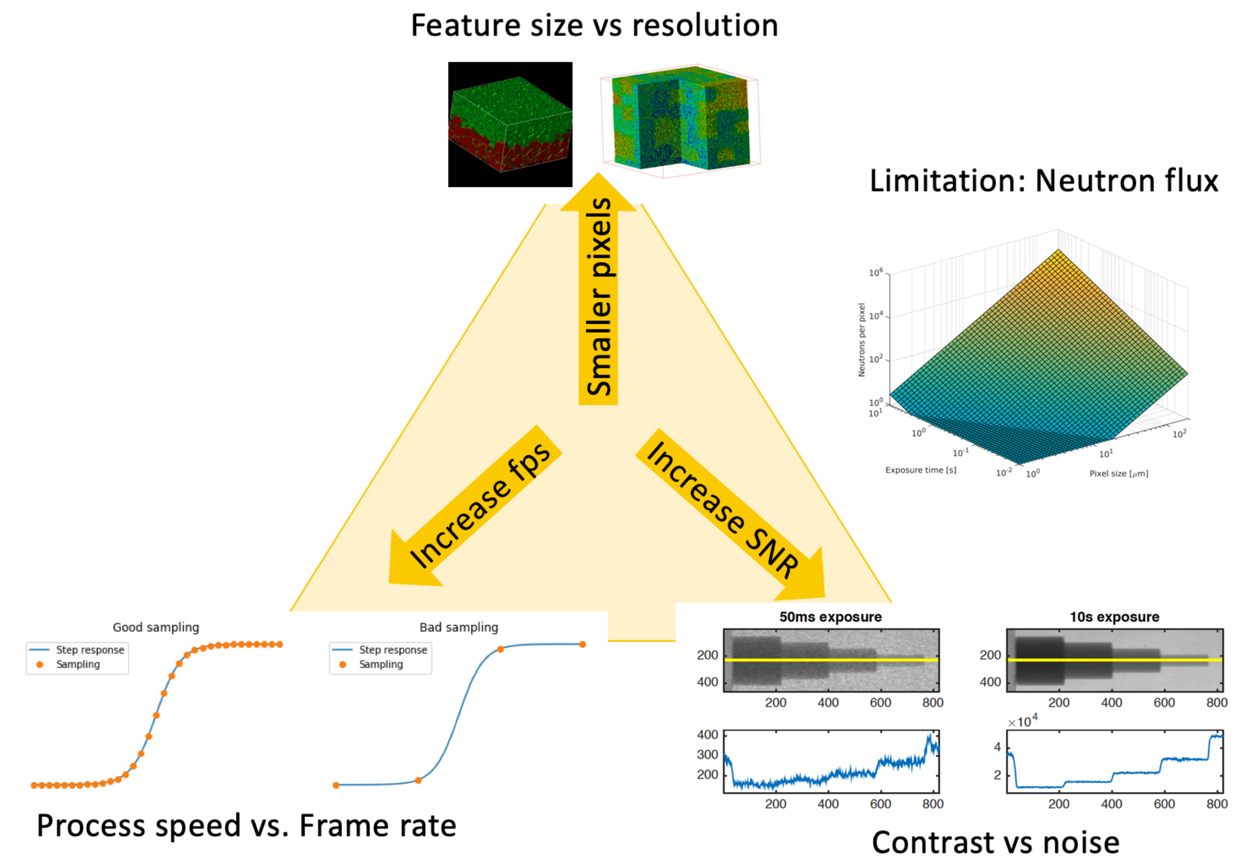


Fig. 6: Factors contributing to the deciding how to perform a dynamic experiment.

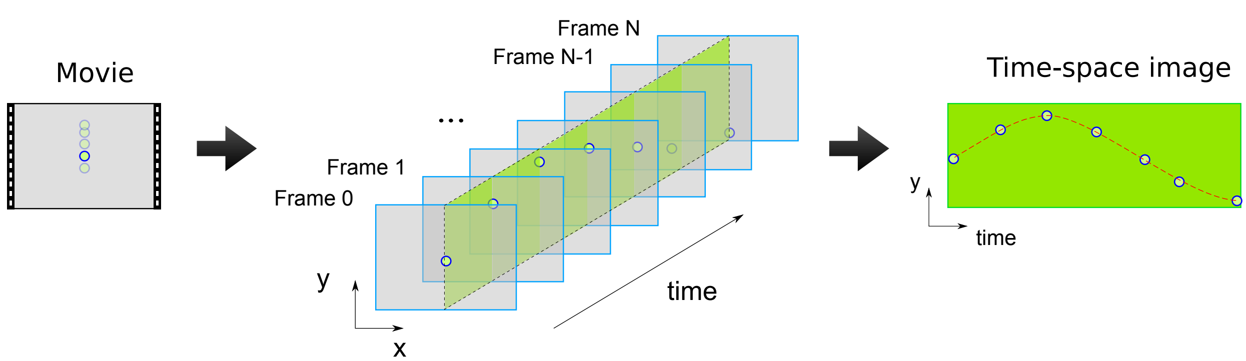


Fig. 7: Extracting a yt-image from a movie.

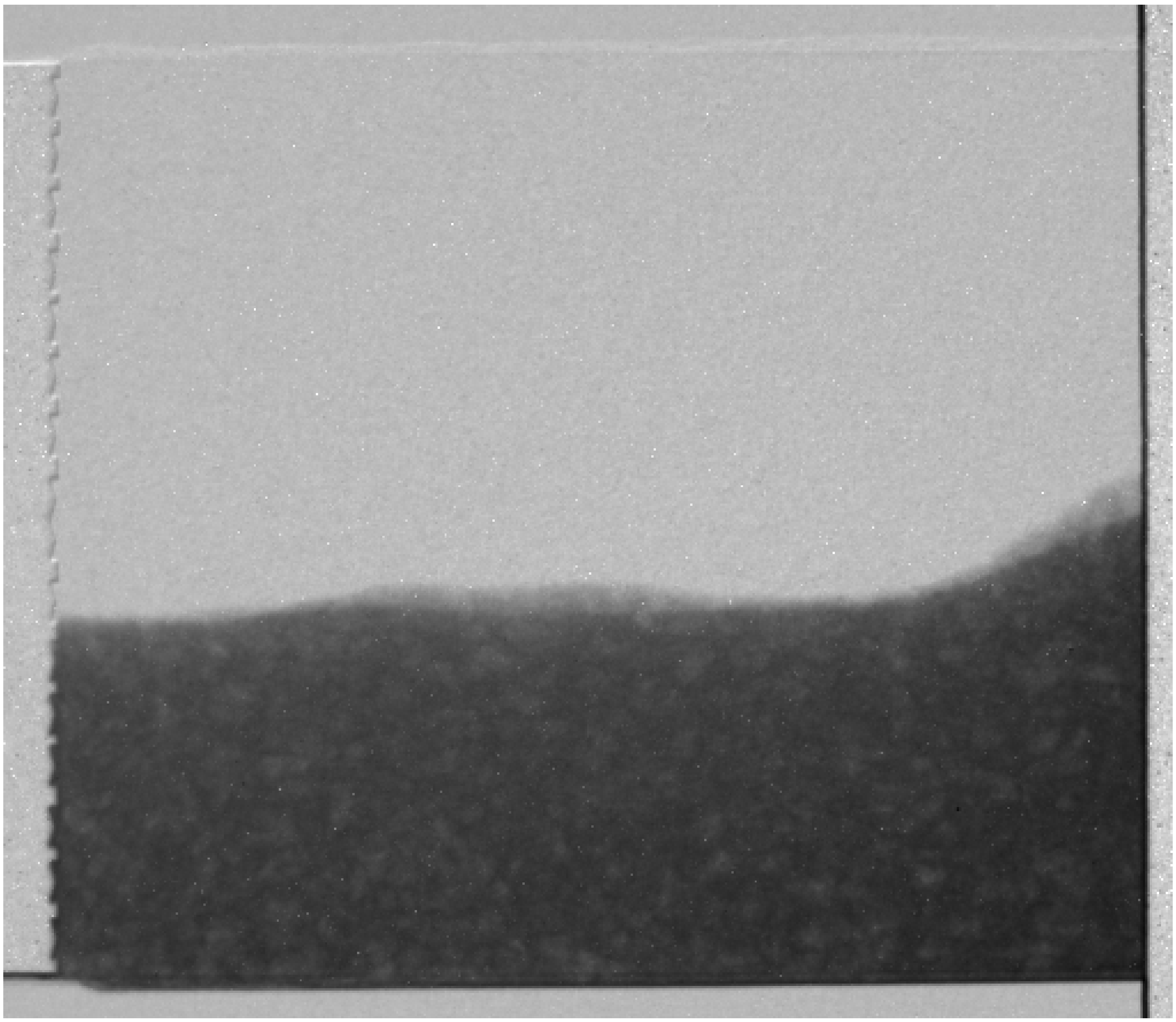


Fig. 8: A frame from the capillary rise movie.

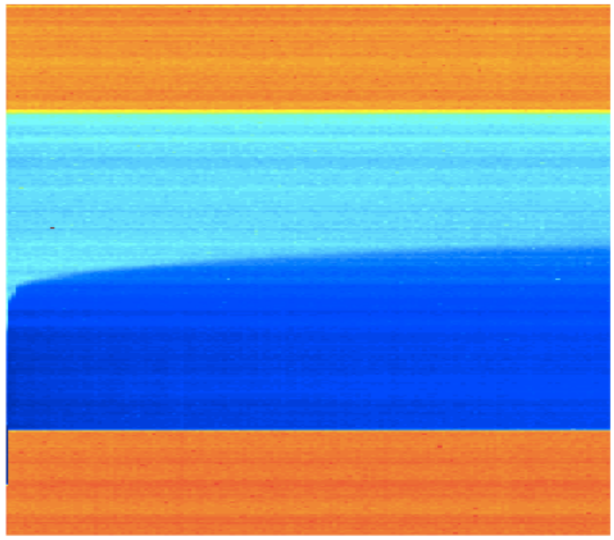


Fig. 9: An extracted yt-slice.

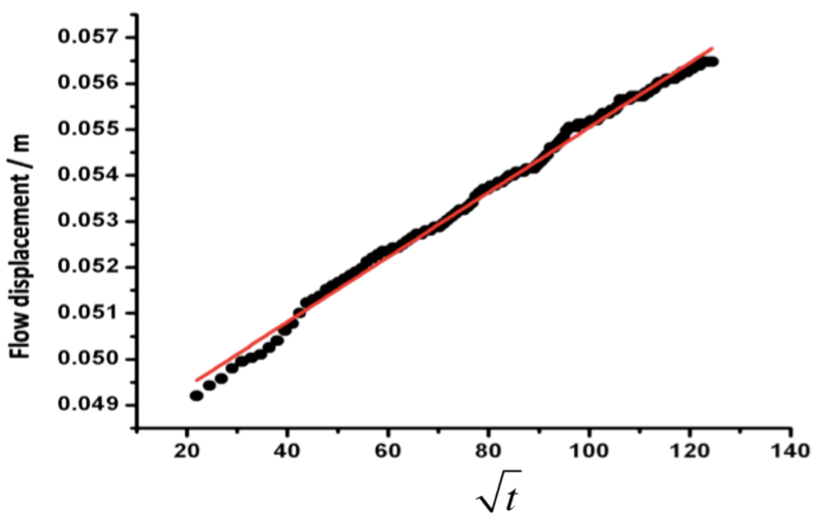


Fig. 10: Front positions follow Washbourne's equation.

0.4.3 Tuning experiment by simulation

In many cases:

- experimental data or setup is inherited
- little can be done about the design,

When there is still the opportunity to tune the experiment

simulations provide a powerful tool for tuning and balancing a large number parameters

Validation

Simulations also provide the ability to pair post-processing to the experiments and determine the limits of tracking.

0.4.4 What do we start with?

Going back to our original cell image

1. We have been able to *get rid of the noise* in the image and *find all the cells* (**lecture 2-4**)
2. We have analyzed the shape of the cells using the shape tensor (**lecture 5**)
3. We even *separated cells* joined together using Watershed (**lecture 6**)
4. We have created even more *metrics characterizing the distribution* (**lecture 7**)

We have at least a few samples (or different regions), large number of metrics and an almost as large number of parameters to *tune*

How do we do something meaningful with it?

0.4.5 Pixel/Voxel-based Methods

- Cross Correlation
- DIC
- DIC + Physics
- Affine Transforms
- Non-rigid transform

Keypoint Detection

- Corner Detectors
- SIFT/SURF
- Tracking from Keypoints

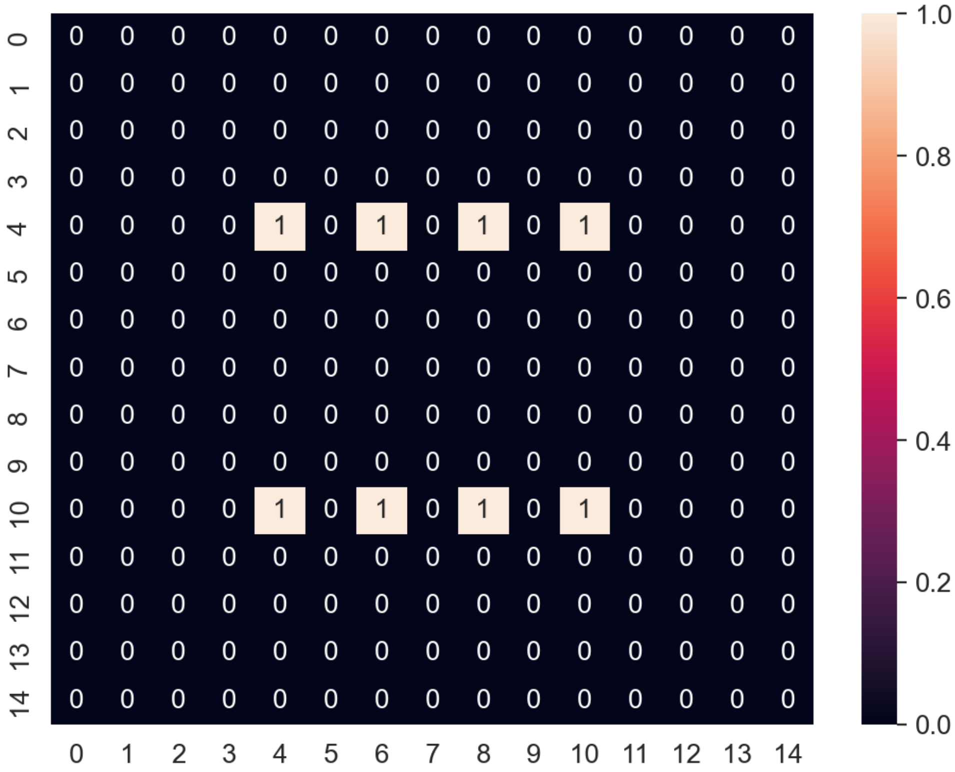
0.4.6 General Problems

- Thickness - Lung Tissue
- Curvature - Metal Systems
- Two Point Correlation - Volcanic Rock

0.5 A basic simulation

We start with a starting image with a number of circles on a plane

```
import numpy as np
import matplotlib.pyplot as plt
from skimage.io import imread
%matplotlib inline
xx, yy = np.meshgrid(np.linspace(-1.5, 1.5, 15),
                     np.linspace(-1.5, 1.5, 15))

N_DISK_ROW = 2
N_DISK_COL = 4
DISK_RAD = 0.15
disk_img = np.zeros(xx.shape, dtype=int)
for x_cent in 0.7*np.linspace(-1, 1, N_DISK_COL):
    for y_cent in 0.7*np.linspace(-1, 1, N_DISK_ROW):
        c_disk = np.sqrt(np.square(xx-x_cent)+np.square(yy-y_cent)) < DISK_RAD
        disk_img[c_disk] = 1
fig, ax1 = plt.subplots(1, 1)

sns.heatmap(disk_img, annot=True, fmt='d', ax=ax1);
```



0.5.1 Create a series of moving "cells"

```
from matplotlib.animation import FuncAnimation
from IPython.display import HTML
fig, c_ax = plt.subplots(1, 1, figsize=(10, 10), dpi=200)

s_img = disk_img.copy()
img_list = [s_img]
for i in range(4):
    s_img = np.roll(s_img, -1, axis=1)
    s_img = np.roll(s_img, -1, axis=0)
    img_list += [s_img]

def update_frame(i):
    plt.cla()
    sns.heatmap(img_list[i],annot=True,
                fmt="d",cmap='nipy_spectral',
                ax=c_ax,cbar=False,
                vmin=0,vmax=1)
    c_ax.set_title('Iteration #{}'.format(i+1))

# write animation frames
anim_code = FuncAnimation(fig,
                          update_frame,
                          frames=len(img_list),
                          interval=1000, repeat_delay=2000)
anim_code.save('movies/moving_cells.mp4', fps=2, extra_args=['-vcodec', 'libx264'])
plt.close('all')
```

0.5.2 Analysis

The analysis of the series requires the following steps:

1. Threshold
2. Component Label
3. Shape Analysis (label, position, area, and frame id)
4. Distribution Analysis

… and to put all in a data frame

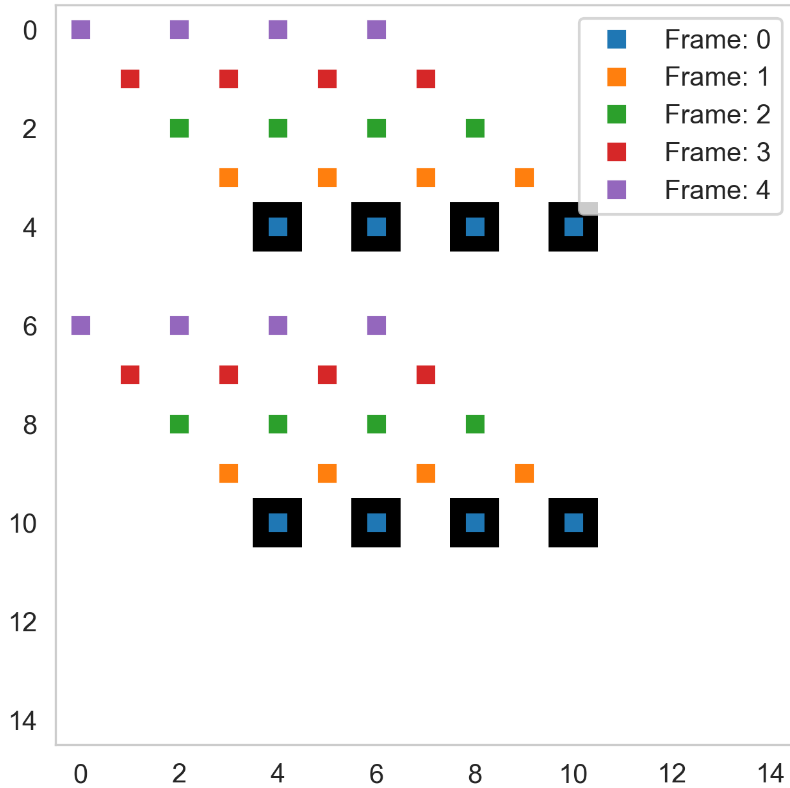
```
all_objs = []
for frame_idx, c_img in enumerate(img_list):        # For each time frame
    lab_img = label(c_img > 0)                       # Label the items
    for c_obj in regionprops(lab_img):               # Put region properties for each␣
↪object of the time frame
        all_objs += [dict(label     = int(c_obj.label),
                          y         = c_obj.centroid[0],
                          x         = c_obj.centroid[1],
                          area      = c_obj.area,
                          frame_idx = frame_idx)]

all_obj_df = pd.DataFrame(all_objs)                  # Create a Pandas data frame with␣
↪all the properties
all_obj_df.head(5)                                   # Look at the first five rows of␣
↪the data frame
```

```
   label     y     x  area  frame_idx
0      1   4.0   4.0   1.0          0
1      2   4.0   6.0   1.0          0
2      3   4.0   8.0   1.0          0
3      4   4.0  10.0   1.0          0
4      5  10.0   4.0   1.0          0
```

0.5.3 Looking at the positions of the items in all frames

Here, we look at the positions of the items found in each time frame. The information comes from the data frame we just created. We only use position and frame index here.

```
fig, c_ax = plt.subplots(1, 1, figsize=(5, 5), dpi=150)
c_ax.imshow(disk_img, cmap='bone_r')
for frame_idx, c_rows in all_obj_df.groupby('frame_idx'):
    c_ax.plot(c_rows['x'], c_rows['y'], 's', label='Frame: %d' % frame_idx)
c_ax.legend();
```

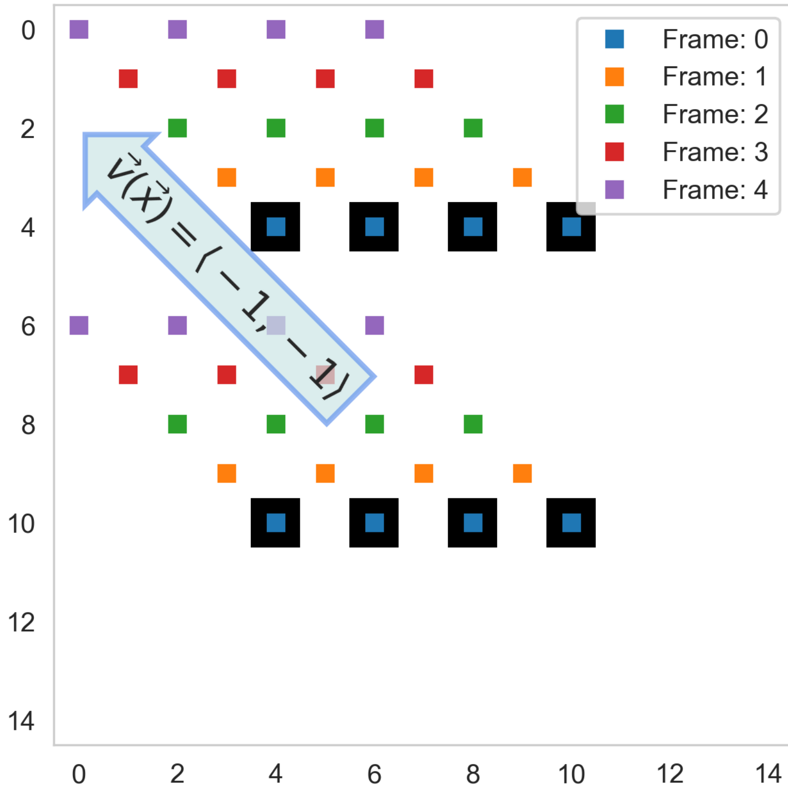


0.6 Describing Motion

We can describe the motion in the above example with a simple vector

$$\vec{v}(\vec{x}) = \langle -1, -1 \rangle$$

```
fig, c_ax = plt.subplots(1, 1, figsize=(5, 5), dpi=150)
c_ax.imshow(disk_img, cmap='bone_r')
for frame_idx, c_rows in all_obj_df.groupby('frame_idx'):
    c_ax.plot(c_rows['x'], c_rows['y'], 's', label='Frame: %d' % frame_idx)
c_ax.legend();

bbox_props = dict(boxstyle="larrow", fc=(0.8, 0.9, 0.9), ec='cornflowerblue', lw=2,
↪alpha=0.7)
t = c_ax.text(3,5, r"$ \vec{v}(\vec{x})=\langle -1,-1 \rangle $", ha="center", va=
↪"center", rotation=-45,
            size=15,
            bbox=bbox_props)
```



This is a very simple case of motion as it only follows a straight line towards the upper left corner. Motion in real experiments are much more complicated and it can also ambiguous which item is which when the trajectories are crossing.

0.7 Scoring Tracking

In the following examples we will use simple metrics for scoring fits where the objects are matched and the number of misses is counted.

There are a number of more sensitive scoring metrics which can be used, by finding the best submatch for a given particle since the number of matches and particles does not always correspond.

See the papers at the beginning for more information

0.8 Tracking methods

While there exist a number of different methods and complicated approaches for tracking.

For experimental design it is best to start with the (Occam's razor)

- simplest
- easiest understood method.

The limits of this can be found and components added as needed until it is possible to realize the experiment

---

If a dataset can only be analyzed with a multiple-hypothesis testing neural network model then it might not be so reliable

0.9 Tracking using Nearest Neighbor

We then return to *nearest neighbor* which means we track

- a point ($\vec{P}_0$) from an image (I_0) at t_0
- to a point ($\vec{P}_1$) in image (I_1) at t_1

by

$$\vec{P}_1 = \text{argmin}(||\vec{P}_0 - \vec{y}|| \; \forall \vec{y} \in I_1)$$

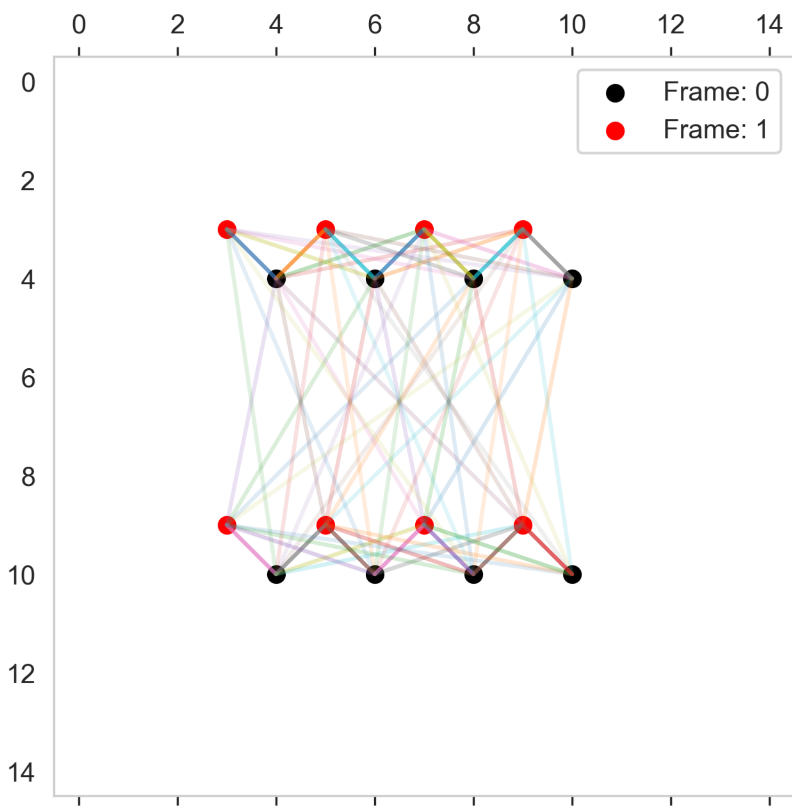
0.10 Distances between objects

The plot below shows the distances from each object to all other objects. Without additional information like

- we are looking for the nearest neighbor
- or the expected displacement is bounded within some limits.
- shape characteristics of the items

It is impossible to tell which is the matching object between frame 0 and 1.

```
frame_0 = all_obj_df[all_obj_df['frame_idx'].isin([0])]
frame_1 = all_obj_df[all_obj_df['frame_idx'].isin([1])]
fig, c_ax = plt.subplots(1, 1, figsize=(5, 5), dpi=150)
c_ax.matshow(1 < disk_img , cmap='gist_yarg')
c_ax.scatter(frame_0['x'], frame_0['y'], c='black', label='Frame: 0')
c_ax.scatter(frame_1['x'], frame_1['y'], c='red', label='Frame: 1')
dist_df_list = []
for _, row_0 in frame_0.iterrows():
    for _, row_1 in frame_1.iterrows():
        seg_dist = np.sqrt(np.square(row_0['x']-row_1['x']) +
                           np.square(row_0['y']-row_1['y']))
        c_ax.plot([row_0['x'], row_1['x']],
                  [row_0['y'], row_1['y']], '-', alpha=1/seg_dist)
        dist_df_list += [dict(x0=row_0['x'],
                              y0=row_0['y'],
                              lab0=int(row_0['label']),
                              x1=row_1['x'],
                              y1=row_1['y'],
                              lab1=int(row_1['label']),
                              dist=seg_dist)]
c_ax.legend();
```



0.10.1 Put the distances in a data frame

```
dist_df = pd.DataFrame(dist_df_list)
dist_df.head(5)
```

	x0	y0	lab0	x1	y1	lab1	dist
0	4.0	4.0	1	3.0	3.0	1	1.414214
1	4.0	4.0	1	5.0	3.0	2	1.414214
2	4.0	4.0	1	7.0	3.0	3	3.162278
3	4.0	4.0	1	9.0	3.0	4	5.099020
4	4.0	4.0	1	3.0	9.0	5	5.099020

```
from matplotlib.animation import FuncAnimation
from IPython.display import HTML

fig, c_ax = plt.subplots(1, 1, figsize=(5, 5), dpi=150)
c_ax.matshow(disk_img > 1, cmap='gist_yarg')
c_ax.scatter(frame_0['x'], frame_0['y'], c='black', label='Frame: 0')
c_ax.scatter(frame_1['x'], frame_1['y'], c='red', label='Frame: 1')

def update_frame(i):
    # plt.cla()
    c_rows = dist_df.query('lab0==%d' % i)
    for _, c_row in c_rows.iterrows():
        c_ax.quiver(c_row['x0'], c_row['y0'],
```

(continues on next page)

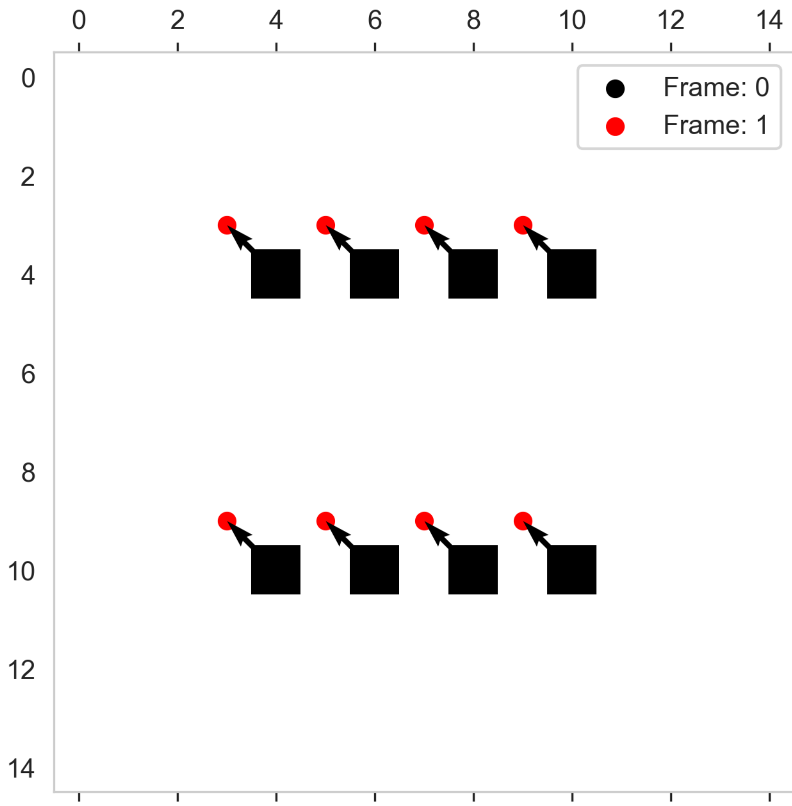
(continued from previous page)

```
                    c_row['x1']-c_row['x0'],
                    c_row['y1']-c_row['y0'], scale=1.0, scale_units='xy', angles='xy',
                    alpha=1/c_row['dist'])
    c_ax.set_title('Point #{}'.format(i+1))



# write animation frames
anim_code = FuncAnimation(fig,
                          update_frame,
                          frames=np.unique(dist_df['lab0']),
                          interval=1000,
                          repeat_delay=2000)

anim_code.save('movies/neighbor_distances.mp4', fps=2, extra_args=['-vcodec', 'libx264
↪'])
plt.close('all')
```

With the nearest neighbor criterion we still have the problem because most items have two possible next position.

Looking at the nearest points found

```
fig, c_ax = plt.subplots(1, 1, figsize=(5, 5), dpi=150)
c_ax.matshow(disk_img >= 1,cmap='gist_yarg')
c_ax.scatter(frame_0['x'], frame_0['y'], c='black', label='Frame: 0')
c_ax.scatter(frame_1['x'], frame_1['y'], c='red', label='Frame: 1')
for _, c_rows in dist_df.groupby('lab0'):
    _, c_row = next(c_rows.sort_values('dist').iterrows())
    c_ax.quiver(c_row['x0'], c_row['y0'],
                c_row['x1']-c_row['x0'],
                c_row['y1']-c_row['y0'],
                scale=1.0, scale_units='xy', angles='xy')

c_ax.legend();
```



Tracking in all frames

This tracking seems to be success full. The reason is that we were lucky that the direction is -1,-1. On the other hand, it is relatively rare that you have perfect positioning of the items and therefore this ambiguity doesn't happen very often.

```
from matplotlib.animation import FuncAnimation
from IPython.display import HTML

fig, c_ax = plt.subplots(1, 1, figsize=(5, 5), dpi=150)
c_ax.matshow(disk_img >= 1,cmap='gist_yarg')


def draw_timestep(i):
    # plt.cla()
    frame_0 = all_obj_df[all_obj_df['frame_idx'].isin([i])]
    frame_1 = all_obj_df[all_obj_df['frame_idx'].isin([i+1])]
    c_ax.scatter(frame_0['x'], frame_0['y'], c='black', label='Frame: %d' % i)
    c_ax.scatter(frame_1['x'], frame_1['y'],
                 c='red', label='Frame: %d' % (i+1))
    dist_df_list = []
    for _, row_0 in frame_0.iterrows():
        for _, row_1 in frame_1.iterrows():
            dist_df_list += [dict(x0=row_0['x'],
                                  y0=row_0['y'],
                                  lab0=int(row_0['label']),
                                  x1=row_1['x'],
                                  y1=row_1['y'],
```

(continues on next page)

(continued from previous page)

```
                                      lab1=int(row_1['label']),
                                      dist=np.sqrt(
                                          np.square(row_0['x']-row_1['x']) +
                                          np.square(row_0['y']-row_1['y'])))]
     dist_df = pd.DataFrame(dist_df_list)
     for _, c_rows in dist_df.groupby('lab0'):
         _, best_row = next(c_rows.sort_values('dist').iterrows())
         c_ax.quiver(best_row['x0'], best_row['y0'],
                     best_row['x1']-best_row['x0'],
                     best_row['y1']-best_row['y0'],
                     scale=1.0, scale_units='xy', angles='xy')
     c_ax.set_title('Frame #{}'.format(i+1))


# write animation frames
anim_code = FuncAnimation(fig,
                          draw_timestep,
                          frames=all_obj_df['frame_idx'].max(),
                          interval=1000,
                          repeat_delay=2000)

anim_code.save('movies/tracking_all_frames.mp4', fps=2, extra_args=['-vcodec',
 ↪'libx264'])
plt.close('all')
```

0.11 Key Points (or feature points)

- Tracking and registration using the full data set is time demaning.
- We can detect feature points in an image and use them to make a registration.

The key points located at characteristic positions around the obejct and their positions relative to each other are invariant to translation and rotation. The points will still be at the same features when the object is skewed, but then they obtain new relative positions compared to the original.

0.11.1 Identifying key points

We first focus on the detection of points.

- Corners are of most interest.

Many methods have been proposed to detect corners. A Harris corner detector helps us here:

The Harris corner detector uses the structure tensor to identify corner points in the image. In the checkerboard example below we identify the corners using the Harris corner detector. In the corner feature image in the middle you see that the corners/crossings provide a strong response. The feature image is thresholded to provide the corner points, which are plotted in the panel to the right.

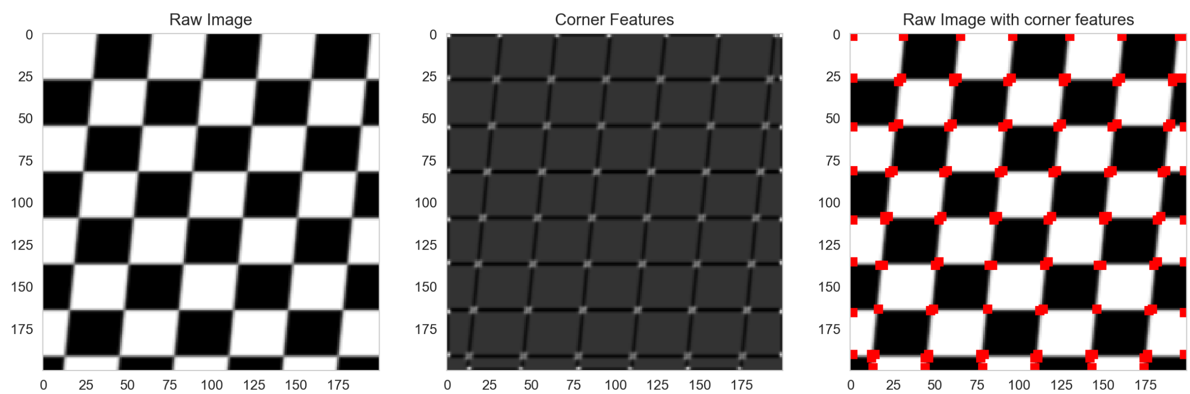
```
tform        = AffineTransform(scale=(1.3, 1.1), rotation=0, shear=0.1,translation=(0,␣
 ↪0))
image        = warp(data.checkerboard(), tform.inverse, output_shape=(200, 200))
```

(continues on next page)

(continued from previous page)

```
corners     = corner_harris(image)
peak_coords = corner_peaks(corners,threshold_rel=0)
```

```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5))
ax1.imshow(image,interpolation='none',cmap='gray'); ax1.set_title('Raw Image')

ax2.imshow(corners,interpolation='none',cmap='gray'); ax2.set_title('Corner Features')

ax3.imshow(image,interpolation='none',cmap='gray'); ax3.set_title('Raw Image with␣
↪corner features')
ax3.plot(peak_coords[:, 1], peak_coords[:, 0], 's',color='red');
```

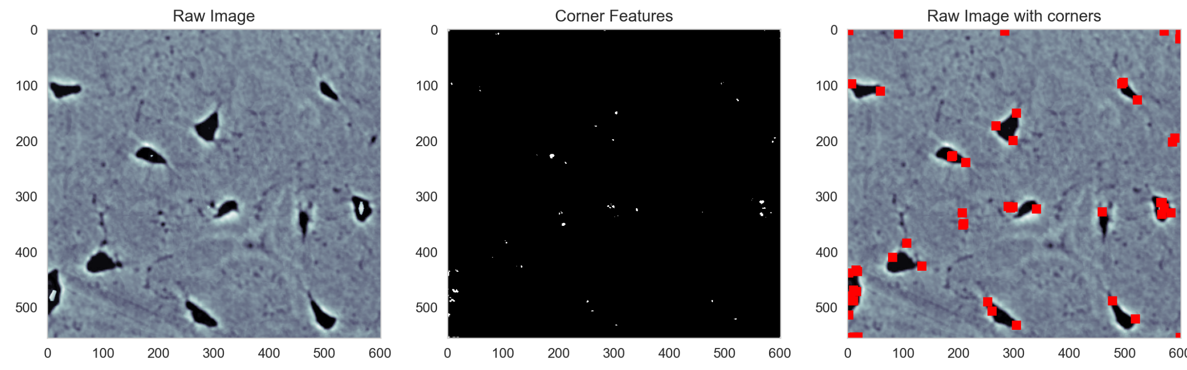


0.11.2 Let's try the corner detection on real data

We return to the bone image we have use many times before and test the Harris detector to identify key points in a natural image.

```
full_img    = imread("figures/bonegfiltslice.png").mean(axis=2)
corners     = corner_harris(full_img)
peak_coords = corner_peaks(corners,threshold_rel=0.002)
```

```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5))
ax1.imshow(full_img,cmap='bone')
ax1.set_title('Raw Image')
ax2.imshow(1.5e7<corners, cmap='gray', interpolation='none'),
ax2.set_title('Corner Features')


ax3.imshow(full_img,cmap='bone'),
ax3.set_title('Raw Image with corners')
ax3.plot(peak_coords[:, 1], peak_coords[:, 0], 'rs');
```



0.12 Tracking with Points

Goal: To reducing the tracking efforts

We can use the corner points to track features between multiple frames.

In this sample, we see that they are

- quite stable
- and fixed

on the features.

0.12.1 We need data - a series transformed images

In this example we apply different affine transformations like

- Translation in x and y
- Rotation
- Shear

to see how well the Harris detector copes with these. The animation clearly shows that the corners are found. It can however happen that there are various number of pixels allocated to each corner point. This can be pruned using morphological algorihms if needed.

```
from matplotlib.animation import FuncAnimation
from IPython.display import HTML
fig, c_ax = plt.subplots(1, 1, figsize=(5, 5), dpi=100)


def update_frame(i):
    c_ax.cla()
    tform = AffineTransform(scale=(1.3+i/20, 1.1-i/20), rotation=-i/10, shear=i/20,
                            translation=(0, 0))
    image = warp(data.checkerboard(), tform.inverse, output_shape=(200, 200))
    c_ax.imshow(image,interpolation='none')
    peak_coords = corner_peaks(corner_harris(image),threshold_rel=0.1)
    c_ax.plot(peak_coords[:, 1], peak_coords[:, 0], 's',color='lightgreen')
```

(continues on next page)

(continued from previous page)

```
# write animation frames
anim_code = FuncAnimation(fig,
                          update_frame,
                          frames=np.linspace(0, 5, 10),
                          interval=1000,
                          repeat_delay=2000)


anim_code.save('movies/affine_transformation.mp4', fps=2, extra_args=['-vcodec',
 ↪'libx264'])
plt.close('all')
```

0.13 Features and Descriptors

We can move beyond just key points to key points and feature vectors (called descriptors) at those points.

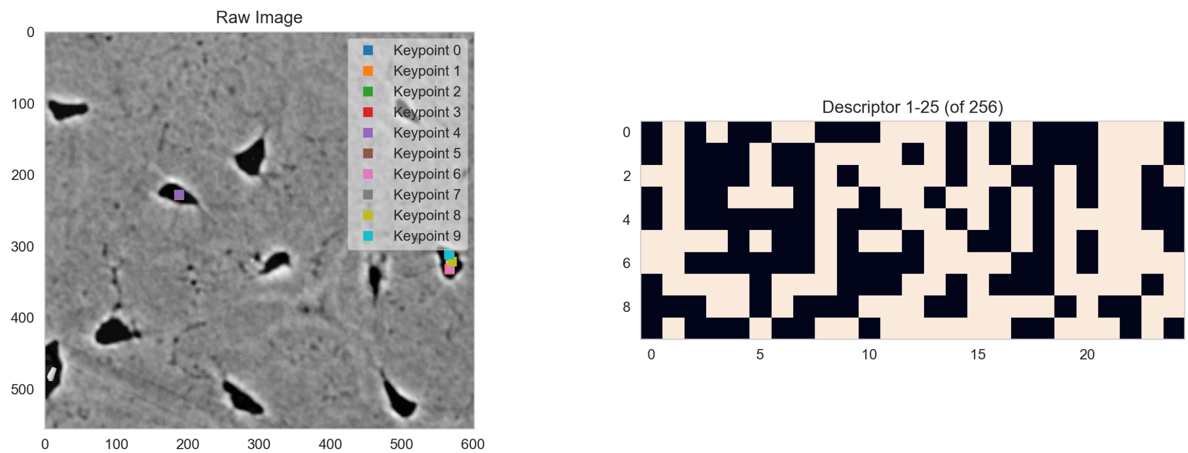
A descriptor is a vector that describes a given keypoint uniquely.

A common descriptor is computed using the ORB algorithm based on

- FAST keypoint detector (**F** eatures from **a** ccelerated **s** egment **t** est)
- BRIEF descriptor (**B** inary **R** obust **I** ndependent **E** lementary **F** eatures)

This will be demonstrated using two methods in the following notebook cells…

```
from skimage.feature import ORB
full_img = imread("figures/bonegfiltslice.png").mean(axis=2)
orb_det  = ORB(n_keypoints=10)
det_obj  = orb_det.detect_and_extract(full_img)

fig, (ax3, ax5) = plt.subplots(1, 2, figsize=(15, 5))
ax3.imshow(full_img, cmap='gray')
ax3.set_title('Raw Image')
for i in range(orb_det.keypoints.shape[0]):
    ax3.plot(orb_det.keypoints[i, 1], orb_det.keypoints[i,
                                                         0], 's', label='Keypoint {}'.
 ↪format(i))
ax5.imshow(np.stack([x[:25] for x in orb_det.descriptors], 0))
ax5.set_title('Descriptor 1-25 (of 256)')
ax3.legend(facecolor='white', framealpha=0.5);
```



0.13.1 Defining a supporting function to show the matches

This is a support function used to display the matched feature points between two images.

```python
from skimage.feature import match_descriptors, plot_matches
import matplotlib.pyplot as plt


def show_matches(img1, img2, feat1, feat2):
    matches12 = match_descriptors(feat1['descriptors'],
                                  feat2['descriptors'],
                                  cross_check=True)

    fig, (ax3, ax2) = plt.subplots(1, 2, figsize=(15, 5))

    plot_matches(ax3,
                 img1, img2,
                 feat1['keypoints'], feat1['keypoints'],
                 matches12,
                 keypoints_color='r')
#                  matches_color='b')
    ax3.vlines([img1.shape[1]-1],ymin=0,ymax=img1.shape[0]-1,linewidth=2,color='white')

    ax2.plot(feat1['keypoints'][:, 1],
             feat1['keypoints'][:, 0],
             '.',
             label='Before')

    ax2.plot(feat2['keypoints'][:, 1],
             feat2['keypoints'][:, 0],
             '.', label='After')

    for i, (c_idx, n_idx) in enumerate(matches12):
        x_vec = [feat1['keypoints'][c_idx, 0], feat2['keypoints'][n_idx, 0]]
        y_vec = [feat1['keypoints'][c_idx, 1], feat2['keypoints'][n_idx, 1]]
        dist = np.sqrt(np.square(np.diff(x_vec))+np.square(np.diff(y_vec)))
        alpha = np.clip(50/dist, 0, 1)[0]
```

(continues on next page)

(continued from previous page)

```
        ax2.plot(
            y_vec,
            x_vec,
            'k-',
            alpha=alpha,
            label='Match' if i == 0 else ''
        )

    ax2.legend()

    ax3.set_title(r'{} $\rightarrow$ {}'.format('Before', 'After'));
```

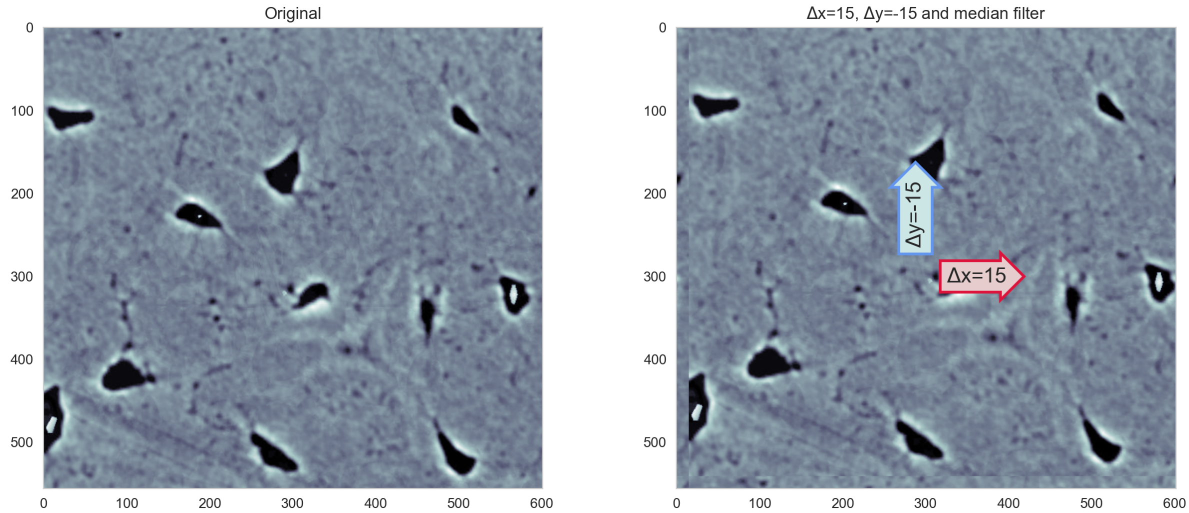
0.13.2 Let's create some data

We make the following modifications to the bone image:

- affine transformation (linear displacement)
- blurring (median filter)

For this example we create a pair of images, where one translated and a smoothing median filter is applied. We will use this image pair to dest different descriptors in the next sections.

```
from skimage.filters import median
full_img = imread("figures/bonegfiltslice.png").mean(axis=2)
full_shift_img = median(
    np.roll(np.roll(full_img, -15, axis=0), 15, axis=1), np.ones((1, 3)))

bw_img    = full_img
shift_img = full_shift_img

fig,ax = plt.subplots(1,2,figsize=(15,6))
ax[0].imshow(bw_img,cmap='bone')
ax[0].set_title('Original')
ax[1].imshow(shift_img,cmap='bone')
ax[1].set_title('$\Delta{}$x=15, $\Delta{}$y=-15 and median filter');

bbox_props = dict(boxstyle="rarrow", fc=(0.9, 0.8, 0.8), ec="crimson", lw=2)
ax[1].text(325, 300, r"$\Delta{}$x=15", ha="left", va="center", rotation=0,
            size=15,
            bbox=bbox_props)

bbox_props = dict(boxstyle="rarrow", fc=(0.8, 0.9, 0.9), ec="cornflowerblue", lw=2)
ax[1].text(275, 225, r"$\Delta{}$y=-15", ha="left", va="center", rotation=90,
            size=15,
            bbox=bbox_props);
```



0.13.3 Features found by the BRIEF descriptor

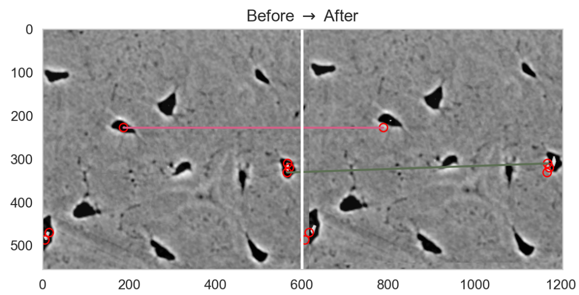
Scan the neighborhood of pixel x, $\mathcal{N}(x)$ to create at local fingerprint

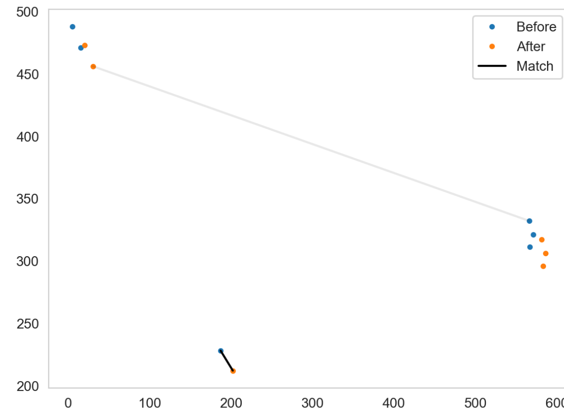
$$\forall_{y \in \mathcal{N}(x)} \begin{cases} 1 & p(x) < p(y) \\ 0 & p(x) \geq p(y) \end{cases}$$

Produces a binary number where each bit is represented by a neighbor pixel.

In this first example we use a combination of the Harris corner detector and BRIEF to create descriptors.

```
from skimage.feature import corner_peaks, corner_harris, BRIEF

def calc_corners(*imgs):
    b = BRIEF()
    for c_img in imgs:
        corner_img = corner_harris(c_img)
        coords = corner_peaks(corner_img, min_distance=5,threshold_rel=0.1)
        b.extract(c_img, coords)
        yield {'keypoints'  : coords,
               'descriptors': b.descriptors}


feat1, feat2 = calc_corners(bw_img, shift_img)
show_matches(bw_img, shift_img, feat1, feat2)
```



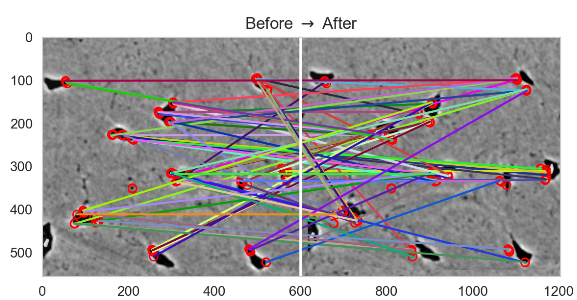


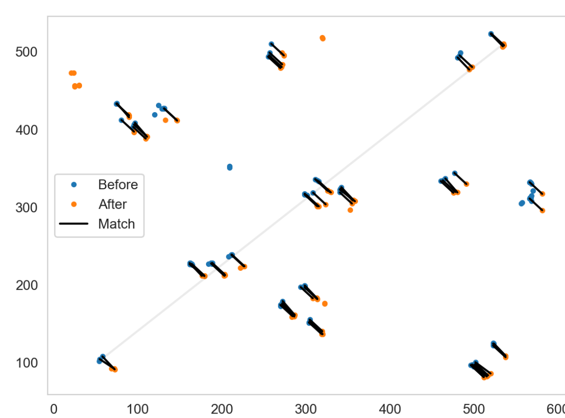
You see two matches in the right panel. One short and the other obviously far away which is less likely to be realistic.

0.13.4 Features found by the ORB descriptor

The ORB descriptor which has a more robust algorithm is able to find many more good pairs between the two time steps.

```
from skimage.feature import ORB, BRIEF, CENSURE


def calc_orb(*imgs):
    descriptor_extractor = ORB(n_keypoints=100)
    for c_img in imgs:
        descriptor_extractor.detect_and_extract(c_img)
        yield {'keypoints': descriptor_extractor.keypoints,
               'descriptors': descriptor_extractor.descriptors}


feat1, feat2 = calc_orb(bw_img, shift_img)
show_matches(bw_img, shift_img, feat1, feat2)
```





0.14 Computing Average Flow

From each of these time steps we can now proceed to compute the average flow.

We can perform this :

- spatially (averaging over regions),
- temporally (averaging over time),
- or spatial-temporally (averaging over regions for every time step)

```
average_field = []
for i in range(all_obj_df['frame_idx'].max()):
    frame_0 = all_obj_df[all_obj_df['frame_idx'].isin([i])]
    frame_1 = all_obj_df[all_obj_df['frame_idx'].isin([i+1])]
    dist_df_list = []
    for _, row_0 in frame_0.iterrows():
        for _, row_1 in frame_1.iterrows():
            dist_df_list += [dict(x0=row_0['x'],
                                  y0=row_0['y'],
                                  lab0=int(row_0['label']),
                                  x1=row_1['x'],
                                  y1=row_1['y'],
                                  lab1=int(row_1['label']),
                                  dist=np.sqrt(
                                      np.square(row_0['x']-row_1['x']) +
                                      np.square(row_0['y']-row_1['y'])))]
    dist_df = pd.DataFrame(dist_df_list)
    for _, c_rows in dist_df.groupby('lab0'):
        _, best_row = next(c_rows.sort_values('dist').iterrows())
        average_field += [dict(frame_idx=i,
                               x=best_row['x0'],
                               y=best_row['y0'],
                               dx=best_row['x1']-best_row['x0'],
                               dy=best_row['y1']-best_row['y0'])]
average_field_df = pd.DataFrame(average_field)
print('Average Flow:\n{}'.format(average_field_df[['dx', 'dy']].mean()))

average_field_df.sample(5)
```

```
Average Flow:
dx   -1.0
dy   -1.0
dtype: float64
```

	frame_idx	x	y	dx	dy
16	2	2.0	2.0	-1.0	-1.0
22	2	6.0	8.0	-1.0	-1.0
15	1	9.0	9.0	-1.0	-1.0
4	0	4.0	10.0	-1.0	-1.0
31	3	7.0	7.0	-1.0	-1.0

Here, create an average field using the dx=-1, dy=-1 moving cells in the all_obj_df. The result is put in a new data frame.

The loop compares items in the current frame with items in the next frame and computes the displacements between all items. A second loop sorts the displacements to find the best combinations.

At the end, the average flow is computed.

0.14.1 Spatially Averaging

To spatially average we first create a grid of values and then interpolate our results onto this grid

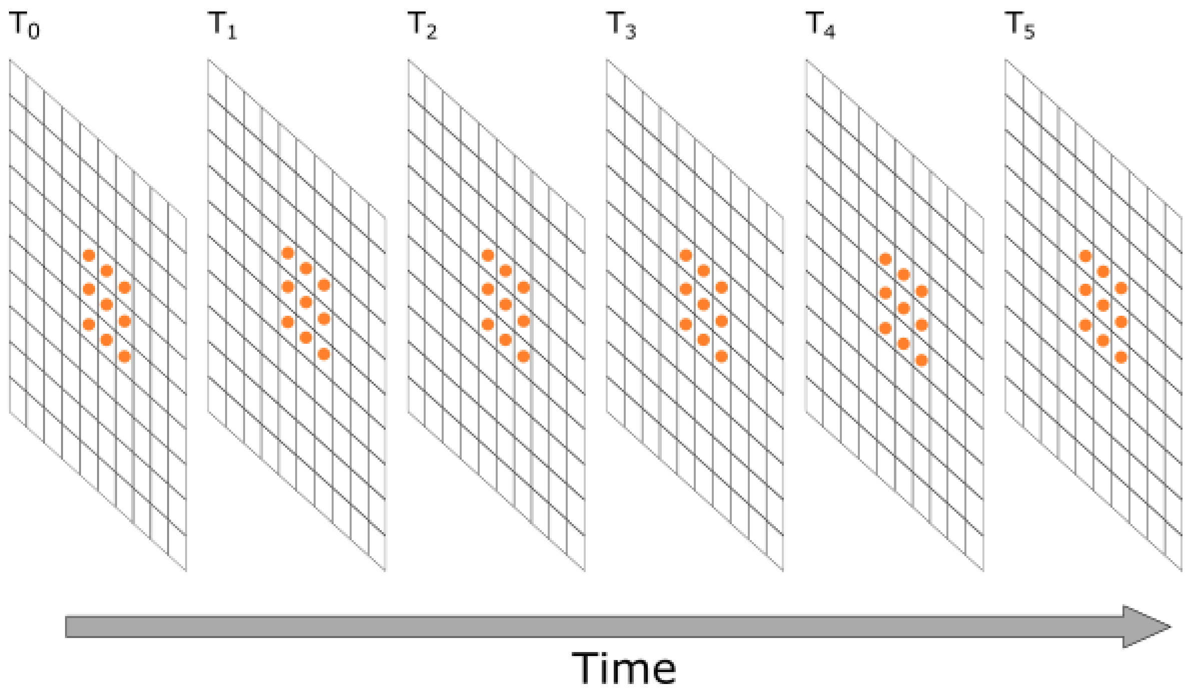


Fig. 1: Spatial averaging.

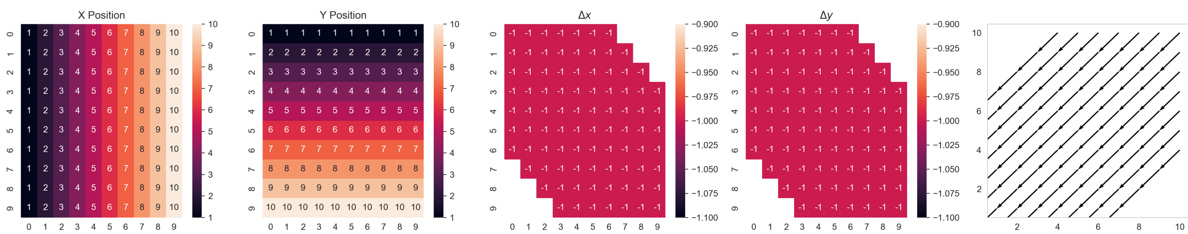
0.14.2 Averaging fields

Vector fields can be very jittery in noisy studies. Spatial averaging can reduce the jitter.

```
from scipy.interpolate import RegularGridInterpolator, LinearNDInterpolator

def img_intp(f):
    def new_f(x, y):
        return np.stack([f(ix, iy) for ix, iy in zip(np.ravel(x), np.ravel(y))], 0).
↪reshape(np.shape(x))
    return new_f


dx_func = img_intp(
    LinearNDInterpolator((average_field_df['x'], average_field_df['y']), average_
↪field_df['dx']))
dy_func = img_intp(
    LinearNDInterpolator((average_field_df['x'], average_field_df['y']), average_
↪field_df['dy']))
dx_func(8, 8), dy_func(8, 8)
```

```
(array(-1.), array(-1.))
```

```
g_x, g_y = np.meshgrid(np.linspace(average_field_df['x'].min(),
                                   average_field_df['x'].max(), 10),
                       np.linspace(average_field_df['y'].min(),
                                   average_field_df['y'].max(), 10))
fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(1, 5, figsize=(24, 4))
sns.heatmap(g_x, ax=ax1, annot=True)
ax1.set_title('X Position')
sns.heatmap(g_y, ax=ax2, annot=True)
ax2.set_title('Y Position')

sns.heatmap(dx_func(g_x, g_y), ax=ax3, annot=True)
ax3.set_title('$\Delta x$')
sns.heatmap(dy_func(g_x, g_y), ax=ax4, annot=True)
ax4.set_title('$\Delta y$')
ax5.quiver(g_x, g_y, dx_func(g_x, g_y), dy_func(g_x, g_y),
           scale=1.0, scale_units='xy', angles='xy');
```



0.14.3 Temporarly Averaging

Here we take the average at each time point

Temporarly averaging on the cell movement

Compute the average displacement for each time frame

```
temp_avg_field = average_field_df[['frame_idx', 'dx', 'dy']].groupby(
    'frame_idx').agg('mean').reset_index()
temp_avg_field
```

	frame_idx	dx	dy
0	0	-1.0	-1.0
1	1	-1.0	-1.0
2	2	-1.0	-1.0
3	3	-1.0	-1.0

```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(18, 4))
ax1.plot(temp_avg_field['frame_idx'], temp_avg_field['dx'], 'rs')
ax1.set_title('$\Delta x$')
ax1.set_xlabel('Timestep')
ax2.plot(temp_avg_field['frame_idx'], temp_avg_field['dy'], 'rs')
ax2.set_title('$\Delta y$')
```

(continues on next page)

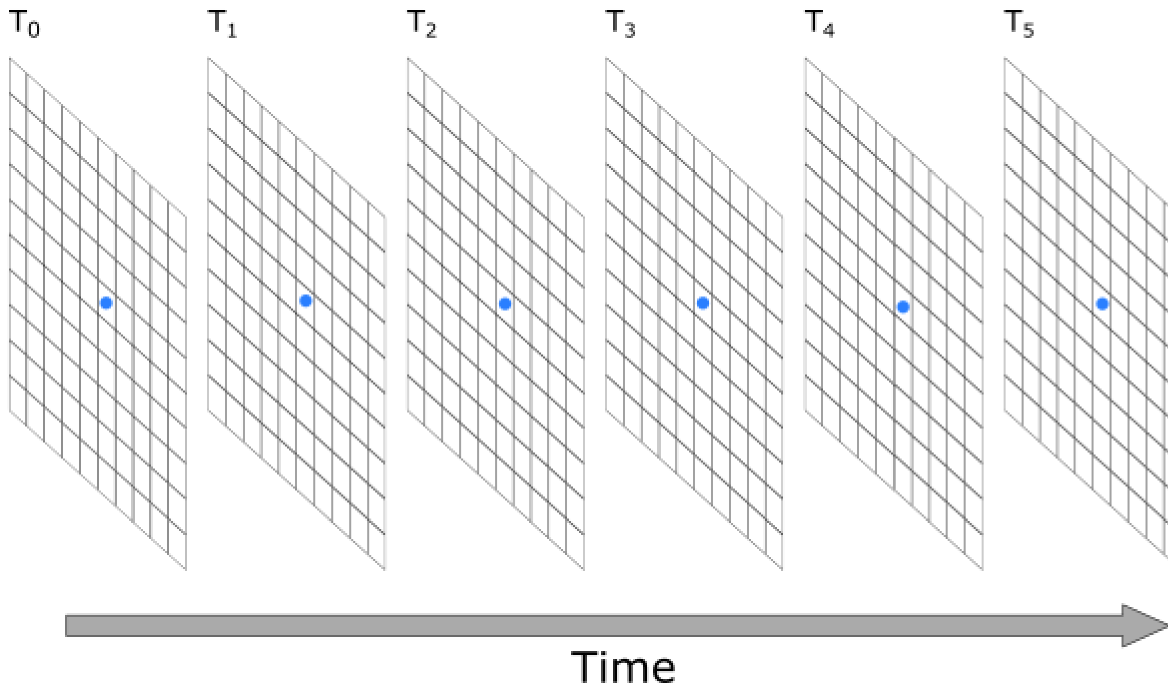
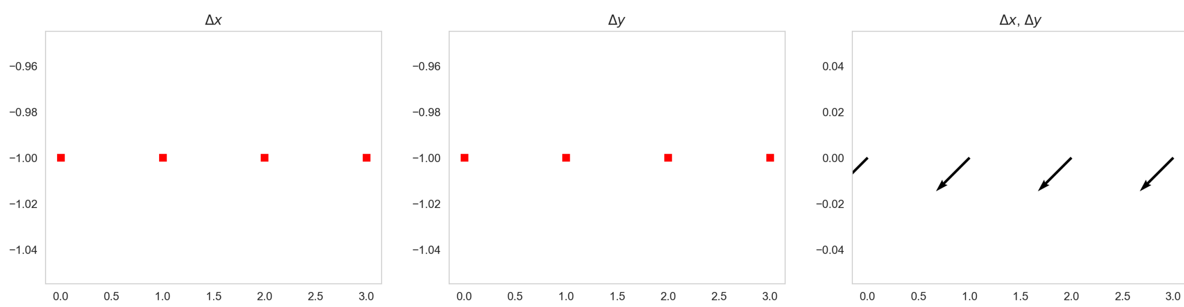


Fig. 2: Temporarily averaging.

(continued from previous page)

```
ax2.set_xlabel('Timestep')
# ax3.quiver(temp_avg_field['dx'], temp_avg_field['dy'],
#             scale=1, scale_units='xy', angles='xy')
ax3.quiver(temp_avg_field['dx'], temp_avg_field['dy'],scale=10)
ax3.set_title('$\Delta x$, $\Delta y$')
ax3.set_xlabel('Timestep');
```



0.14.4 Spatio-temporal Relationship

We can also divide the images into space and time steps

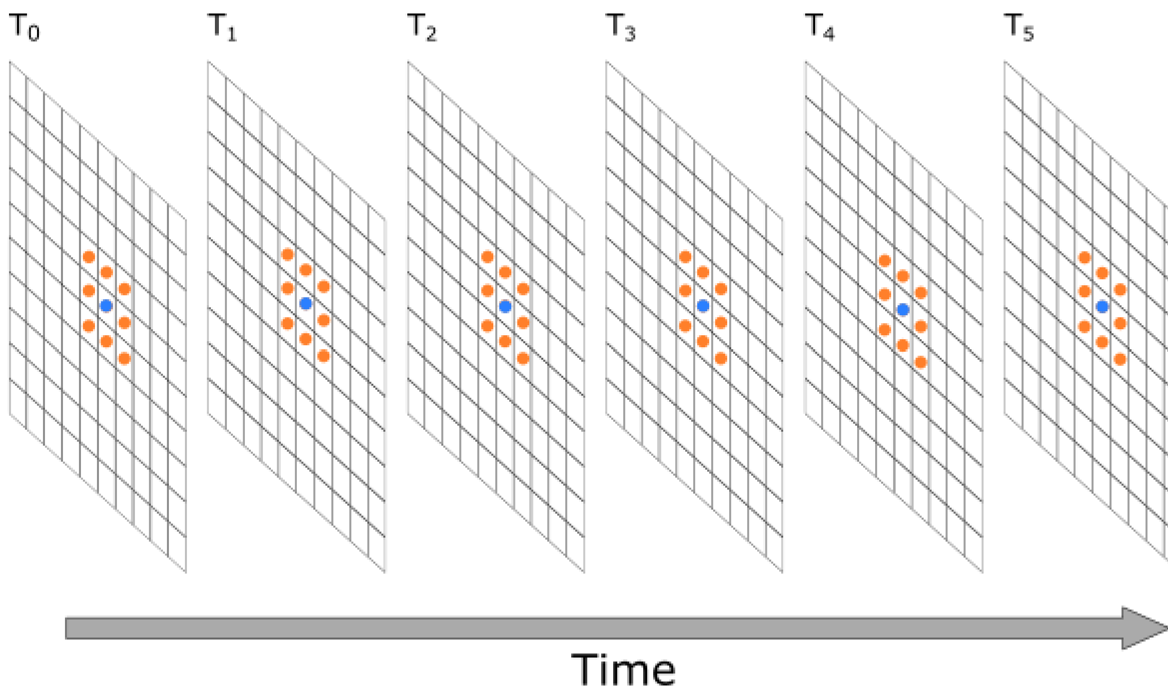


Fig. 3: Spatio-temporal averaging

Spatio-temporal averaging on moving cells

Find all items for a time frame and compute spatial average field.

```
from matplotlib.animation import FuncAnimation
from IPython.display import HTML

g_x, g_y = np.meshgrid(np.linspace(average_field_df['x'].min(),
                                   average_field_df['x'].max(), 4),
                       np.linspace(average_field_df['y'].min(),
                                   average_field_df['y'].max(), 4))

frames = len(sorted(np.unique(average_field_df['frame_idx'])))
fig, m_axs = plt.subplots(2, 3, figsize=(12, 8))
for c_ax in m_axs.flatten():
    c_ax.axis('off')
[(ax1, ax2, _), (ax3, ax4, ax5)] = m_axs


def draw_frame_idx(idx):
    plt.cla()
    c_df = average_field_df[average_field_df['frame_idx'].isin([idx])]
    dx_func = img_intp(LinearNDInterpolator((c_df['x'], c_df['y']), c_df['dx']))
    dy_func = img_intp(LinearNDInterpolator((c_df['x'], c_df['y']), c_df['dy']))
    sns.heatmap(g_x, ax=ax1, annot=False, cbar=False)
```

(continues on next page)

(continued from previous page)

```
    ax1.set_title('Frame %d\nX Position' % idx)
    sns.heatmap(g_y, ax=ax2, annot=False, cbar=False)
    ax2.set_title('Y Position')

    sns.heatmap(dx_func(g_x, g_y).transpose(), ax=ax3, annot=False, cbar=False, fmt=
↪'2.1f')
    ax3.set_title('$\Delta x$')
    sns.heatmap(dy_func(g_x, g_y), ax=ax4, annot=False, cbar=False, fmt='2.1f')
    ax4.set_title('$\Delta y$')
    ax5.quiver(g_x, g_y, dx_func(g_x, g_y), dy_func(g_x, g_y),
               scale=1.0, scale_units='xy', angles='xy')


# write animation frames
anim_code = FuncAnimation(fig,
                          draw_frame_idx,
                          frames=frames,
                          interval=1000,
                          repeat_delay=2000)
anim_code.save('movies/spatiotemporal_avg.mp4', fps=2, extra_args=['-vcodec', 'libx264
↪'])
plt.close('all')
```

0.14.5 Longer Series

We see that this approach becomes problematic when we want to work with longer series

```
import pandas as pd
from skimage.morphology import label
from skimage.measure import regionprops
from matplotlib.animation import FuncAnimation
from IPython.display import HTML
fig, c_ax = plt.subplots(1, 1, figsize=(5, 5), dpi=150)

s_img = disk_img.copy()
img_list = [s_img]
for i in range(8):
    if i % 2 == 0:
        s_img = np.roll(s_img, -2, axis=0)
    else:
        s_img = np.roll(s_img, -1, axis=1)
    img_list += [s_img]

all_objs = []
for frame_idx, c_img in enumerate(img_list):
    lab_img = label(c_img > 0)
    for c_obj in regionprops(lab_img):
        all_objs += [dict(label=int(c_obj.label),
                          y=c_obj.centroid[0],
                          x=c_obj.centroid[1],
                          area=c_obj.area,
                          frame_idx=frame_idx)]
```

(continues on next page)

(continued from previous page)

```
all_obj_df2 = pd.DataFrame(all_objs)
all_obj_df2.head(5)


def update_frame(i):
    plt.cla()
    sns.heatmap(img_list[i],
                annot=True,
                fmt="d",
                cmap='nipy_spectral',
                ax=c_ax,
                cbar=False,
                vmin=0,
                vmax=1)
    c_ax.set_title('Iteration #{}'.format(i+1))


# write animation frames
anim_code = FuncAnimation(fig,
                          update_frame,
                          frames=len(img_list),
                          interval=1000,
                          repeat_delay=2000)

anim_code.save('movies/longseries.mp4', fps=2, extra_args=['-vcodec', 'libx264'])
plt.close('all')
```

The resulting tracking

Tracking this longer series soon get complicated with the amount of direction changes in the scene.

```
from matplotlib.animation import FuncAnimation
from IPython.display import HTML

fig, c_ax = plt.subplots(1, 1, figsize=(8, 8), dpi=200)
c_ax.matshow(disk_img > 1,cmap='gist_yarg')


def draw_timestep(i):
    # plt.cla()
    frame_0 = all_obj_df2[all_obj_df2['frame_idx'].isin([i])]
    frame_1 = all_obj_df2[all_obj_df2['frame_idx'].isin([i+1])]
    c_ax.scatter(frame_0['x'], frame_0['y'], c='black', label='Frame: %d' % i)
    c_ax.scatter(frame_1['x'], frame_1['y'],
                 c='red', label='Frame: %d' % (i+1))
    dist_df_list = []
    for _, row_0 in frame_0.iterrows():
        for _, row_1 in frame_1.iterrows():
            dist_df_list += [dict(x0=row_0['x'],
                                  y0=row_0['y'],
                                  lab0=int(row_0['label']),
                                  x1=row_1['x'],
                                  y1=row_1['y'],
```

(continues on next page)

(continued from previous page)

```
                                          lab1=int(row_1['label']),
                                          dist=np.sqrt(
                                              np.square(row_0['x']-row_1['x']) +
                                              np.square(row_0['y']-row_1['y'])))]
    dist_df2 = pd.DataFrame(dist_df_list)
    for _, c_rows in dist_df2.groupby('lab0'):
        _, best_row = next(c_rows.sort_values('dist').iterrows())
        c_ax.quiver(best_row['x0'], best_row['y0'],
                    best_row['x1']-best_row['x0'],
                    best_row['y1']-best_row['y0'],
                    scale=1.0, scale_units='xy', angles='xy', alpha=0.25)
    c_ax.set_title('Frame #{}'.format(i+1))


# write animation frames
anim_code = FuncAnimation(fig,
                          draw_timestep,
                          frames=all_obj_df['frame_idx'].max(),
                          interval=1000,
                          repeat_delay=2000)
anim_code.save('movies/tracking_result2.mp4', fps=2, extra_args=['-vcodec', 'libx264
↪'])
plt.close('all')
```

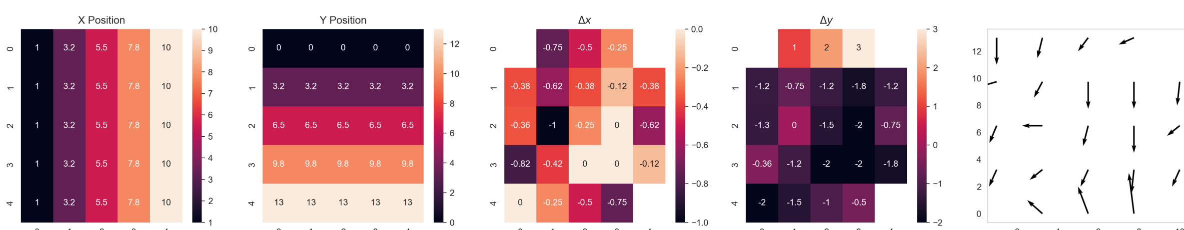
```
from scipy.interpolate import interp2d
average_field = []
for i in range(all_obj_df2['frame_idx'].max()):
    frame_0 = all_obj_df2[all_obj_df2['frame_idx'].isin([i])]
    frame_1 = all_obj_df2[all_obj_df2['frame_idx'].isin([i+1])]
    dist_df_list = []
    for _, row_0 in frame_0.iterrows():
        for _, row_1 in frame_1.iterrows():
            dist_df_list += [dict(x0=row_0['x'],
                                  y0=row_0['y'],
                                  lab0=int(row_0['label']),
                                  x1=row_1['x'],
                                  y1=row_1['y'],
                                  lab1=int(row_1['label']),
                                  dist=np.sqrt(
                                      np.square(row_0['x']-row_1['x']) +
                                      np.square(row_0['y']-row_1['y'])))]
    dist_df = pd.DataFrame(dist_df_list)
    for _, c_rows in dist_df.groupby('lab0'):
        _, best_row = next(c_rows.sort_values('dist').iterrows())
        average_field += [dict(frame_idx=i,
                               x=best_row['x0'],
                               y=best_row['y0'],
                               dx=best_row['x1']-best_row['x0'],
                               dy=best_row['y1']-best_row['y0'])]
average_field_df2 = pd.DataFrame(average_field)
print('Average Flow:')
print(average_field_df2[['dx', 'dy']].mean())
```

(continues on next page)

(continued from previous page)

```
def img_intp(f):
    def new_f(x, y):
        return np.stack([f(ix, iy) for ix, iy in zip(np.ravel(x), np.ravel(y))], 0).
↪reshape(np.shape(x))
    return new_f


dx_func = img_intp(
    LinearNDInterpolator((average_field_df2['x'], average_field_df2['y']), average_
↪field_df2['dx']))
dy_func = img_intp(
    LinearNDInterpolator((average_field_df2['x'], average_field_df2['y']), average_
↪field_df2['dy']))

g_x, g_y = np.meshgrid(np.linspace(average_field_df2['x'].min(),
                                   average_field_df2['x'].max(), 5),
                       np.linspace(average_field_df2['y'].min(),
                                   average_field_df2['y'].max(), 5))
fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(1, 5, figsize=(24, 4))
sns.heatmap(g_x, ax=ax1, annot=True)
ax1.set_title('X Position')
sns.heatmap(g_y, ax=ax2, annot=True)
ax2.set_title('Y Position')

sns.heatmap(dx_func(g_x, g_y), ax=ax3, annot=True)
ax3.set_title('$\Delta x$')
sns.heatmap(dy_func(g_x, g_y), ax=ax4, annot=True)
ax4.set_title('$\Delta y$')
ax5.quiver(g_x, g_y, dx_func(g_x, g_y), dy_func(g_x, g_y),
           scale=1.0, scale_units='xy', angles='xy');
```

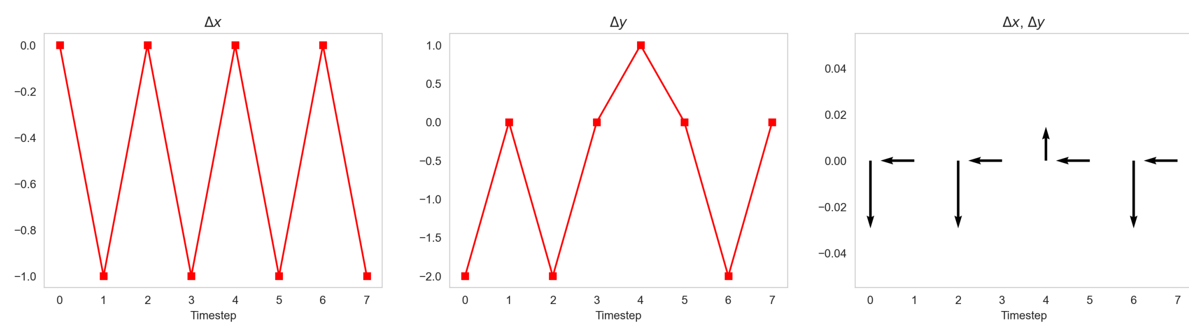
```
Average Flow:
dx    -0.500
dy    -0.625
dtype: float64
```



Temporal averaging of the long series

```
temp_avg_field = average_field_df2[['frame_idx', 'dx', 'dy']].groupby(
    'frame_idx').agg('mean').reset_index()
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(18, 4))
ax1.plot(temp_avg_field['frame_idx'], temp_avg_field['dx'], 'rs-')
ax1.set_title('$\Delta x$')
ax1.set_xlabel('Timestep')
ax2.plot(temp_avg_field['frame_idx'], temp_avg_field['dy'], 'rs-')
ax2.set_title('$\Delta y$')
ax2.set_xlabel('Timestep')
# ax3.quiver(temp_avg_field['dx'], temp_avg_field['dy'],
#            scale=1, scale_units='xy', angles='xy')
ax3.quiver(temp_avg_field['dx'], temp_avg_field['dy'],
           scale=10)
ax3.set_title('$\Delta x$, $\Delta y$')
ax3.set_xlabel('Timestep')
temp_avg_field
```

```
   frame_idx   dx   dy
0          0  0.0 -2.0
1          1 -1.0  0.0
2          2  0.0 -2.0
3          3 -1.0  0.0
4          4  0.0  1.0
5          5 -1.0  0.0
6          6  0.0 -2.0
7          7 -1.0  0.0
```



Spatio-temporal averaging of the longer series

```
from matplotlib.animation import FuncAnimation
from IPython.display import HTML

g_x, g_y = np.meshgrid(np.linspace(average_field_df['x'].min(),
                                   average_field_df['x'].max(), 4),
                       np.linspace(average_field_df['y'].min(),
                                   average_field_df['y'].max(), 4))

frames = len(sorted(np.unique(average_field_df['frame_idx'])))
fig, m_axs = plt.subplots(2, 3, figsize=(14, 10))
for c_ax in m_axs.flatten():
```

(continues on next page)

(continued from previous page)

```
    c_ax.axis('off')
[(ax1, ax2, _), (ax3, ax4, ax5)] = m_axs


def draw_frame_idx(idx):
    plt.cla()
    c_df = average_field_df2[average_field_df2['frame_idx'].isin([idx])]
    dx_func = img_intp(LinearNDInterpolator((c_df['x'], c_df['y']), c_df['dx']))
    dy_func = img_intp(LinearNDInterpolator((c_df['x'], c_df['y']), c_df['dy']))
    sns.heatmap(g_x, ax=ax1, annot=False, cbar=False)
    ax1.set_title('Frame %d\nX Position' % idx)
    sns.heatmap(g_y, ax=ax2, annot=False, cbar=False)
    ax2.set_title('Y Position')

    sns.heatmap(dx_func(g_x, g_y).transpose(), ax=ax3, annot=False, cbar=False, fmt=
↪'2.1f')
    ax3.set_title('$\Delta x$')
    sns.heatmap(dy_func(g_x, g_y), ax=ax4, annot=False, cbar=False, fmt='2.1f')
    ax4.set_title('$\Delta y$')
    ax5.quiver(g_x, g_y, dx_func(g_x, g_y), dy_func(g_x, g_y),
               scale=1.0, scale_units='xy', angles='xy')


# write animation frames
anim_code = FuncAnimation(fig,
                          draw_frame_idx,
                          frames=frames,
                          interval=1000,
                          repeat_delay=2000)
anim_code.save('movies/spacecorrelation_animation.mp4', fps=2, extra_args=['-vcodec',
↪'libx264'])
plt.close('all')
```

Tracking using Trackpy

Someone else did the job for you… trackpy

Tutorial

0.15 Problems with tracking

0.15.1 Random Appearance / Disappearance

Under perfect imaging and experimental conditions objects should not appear and reappear but due to

- Noise
- Limited fields of view / depth of field
- Discrete segmentation approaches
- Motion artifacts

  - Blurred objects often have lower intensity values than still objects

It is common for objects to appear and vanish regularly in an experiment.

0.15.2 Jitter / Motion Noise

Even perfect spherical objects do not move in a straight line:

- The jitter can be seen as a stochastic variable with a random magnitude (a) and angle (b).
- This is then sampled at every point in the field

$$\vec{v}(\vec{x}) = \vec{v}_L(\vec{x}) + ||a||\measuredangle b$$

0.15.3 Limitations of Tracking

We see that visually tracking samples can be difficult and there are a number of parameters which affect the ability for us to clearly see the tracking.

- flow rate
- flow type
- density
- appearance and disappearance rate
- jitter
- particle uniqueness

0.16 How to improve tracking

We have to try to quantify the limits of these parameters for different tracking methods in order to design experiments better.

0.16.1 Acquisition-based Parameters

- Acquisition rate
  - flow rate vs. sampling rate
  - jitter (per frame)
- Resolution
  - density,
  - appearance rate

0.16.2 Experimental Parameters

- Experimental setup (pressure, etc) $\rightarrow$ flow rate/type
- Polydispersity $\rightarrow$ particle uniqueness
- Vibration/temperature $\rightarrow$ jitter
- Mixture $\rightarrow$ density contrast

0.16.3 Basic Simulations

Input flow from simulation

$$\vec{v}(\vec{x}) = \langle 0, 0, 0.05 \rangle + ||0.01||\measuredangle b$$

0.16.4 Designing Experiments

How do simulations help us to design experiments?

- density can be changed by adjusting the concentration of the substances being examined or the field of view
- flow per frame (image velocity) can usually be adjusted by changing pressure or acquisition time
- jitter can be estimated from images

How much is enough?

Difficult to create one number for every experiment

- 5% error in bubble position $\rightarrow$
  - <5% in flow field
  - >20% error in topology
- 5% error in shape or volume $\rightarrow$
  - 5% in distribution or changes
  - $>$ 5% in individual bubble changes
  - $>$ 15% for single bubble strain tensor calculations

0.16.5 Extending Nearest Neighbor

Bijective Requirement

We define $\vec{P}_f$ as the result of performing the nearest neigbhor tracking on $\vec{P}_0 \; \$\vec{P}_f = \text{argmin}(||\vec{P}_0 - \vec{y}|| \forall \vec{y} \in I_1)\$$

We define $\vec{P}_i$ as the result of performing the nearest neigbhor tracking on $\vec{P}_f \; \$\vec{P}_i = \text{argmin}(||\vec{P}_f - \vec{y}|| \forall \vec{y} \in I_0)\$$

We say the tracking is bijective if these two points are the same $\$\vec{P}_i \stackrel{?}{=} \vec{P}_0\$$

Maximum Displacement

$$\vec{P}_1 = \begin{cases} ||\vec{P}_0 - \vec{y}|| < \text{MAXD}, & \text{argmin}(||\vec{P}_0 - \vec{y}|| \forall \vec{y} \in I_1) \\ \text{Otherwise}, & \emptyset \end{cases}$$

0.16.6 Extending Nearest Neighbor (Continued)

Models of movement behavior to support tracking

Prior / Expected Movement

$$\vec{P}_1 = \text{argmin}(||\vec{P}_0 + \vec{v}_{offset} - \vec{y}|| \forall \vec{y} \in I_1)$$

Adaptive Movement

Can then be calculated in an iterative fashion where the offset is the average from all of the $\vec{P}_1 - \vec{P}_0$ vectors. It can also be performed

$$\vec{P}_1 = \text{argmin}(||\vec{P}_0 + \vec{v}_{offset} - \vec{y}|| \forall \vec{y} \in I_1)$$

More advanced models

- Use expected physical model
- Use track derivative to find v_{offset}
- Kalman filters can be used for particle tracking

0.16.7 Beyond Nearest Neighbor

While nearest neighbor

- provides a useful starting tool
- it is not sufficient for truly complicated flows and datasets.

Better Approaches

Multiple Hypothesis Testing

- Nearest neighbor just compares the points between two frames and there is much more information available in most time-resolved datasets.
- This approach allows for multiple possible paths to be explored at the same time and the best chosen only after all frames have been examined

Shortcomings

Merging and Splitting Particles

- The simplicity of the nearest neighbor model does really allow for particles to merge and split (relaxing the bijective requirement allows such behavior, but the method is still not suited for such tracking).
- For such systems a more specific, physically-based is required to encapsulate this behavior.

0.16.8 Voxel-based Approaches

For voxel-based approachs the most common analyses are digital image correlation (or for 3D images digital volume correlation), where the correlation is calculated between two images or volumes.

Standard Image Correlation

Given images $I_0(\vec{x})$ and $I_1(\vec{x})$ at time t_0 and t_1 respectively. The correlation between these two images can be calculated for each $\vec{r}$

$$C_{I_0,I_1}(\vec{r}) = \langle I_0(\vec{x}) I_1(\vec{x}+\vec{r})\rangle$$

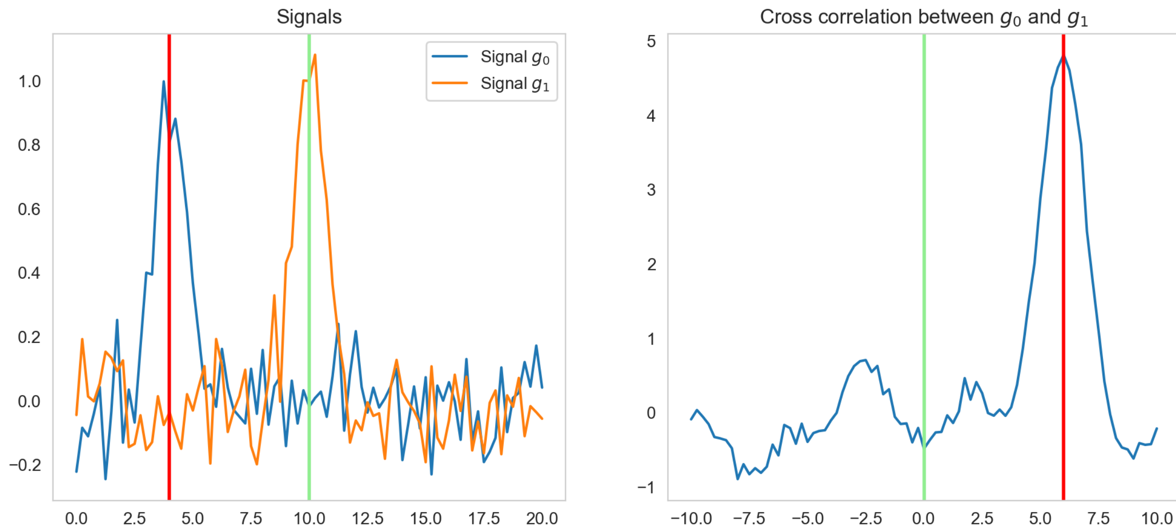
This can also be done in the Fourier space

$$C_{I_0,I_1}(\vec{r}) = \mathcal{F}^{-1}\{\mathcal{F}\{I_0\}\cdot\mathcal{F}\{I_1\}^*\}$$

Correlation in 1D

```
N=81
x=np.linspace(0,20,N)
g0=np.exp(-(x-4)**2)+np.random.normal(0,0.1,size=N)
g1=np.exp(-(x-10)**2)+np.random.normal(0,0.1,size=N)

fg0=np.fft.fft(g0)
fg1=np.fft.fft(g1)
fg0g1=fg1*np.conj(fg0)
cg0g1=np.real(np.fft.ifft(fg0g1))
```

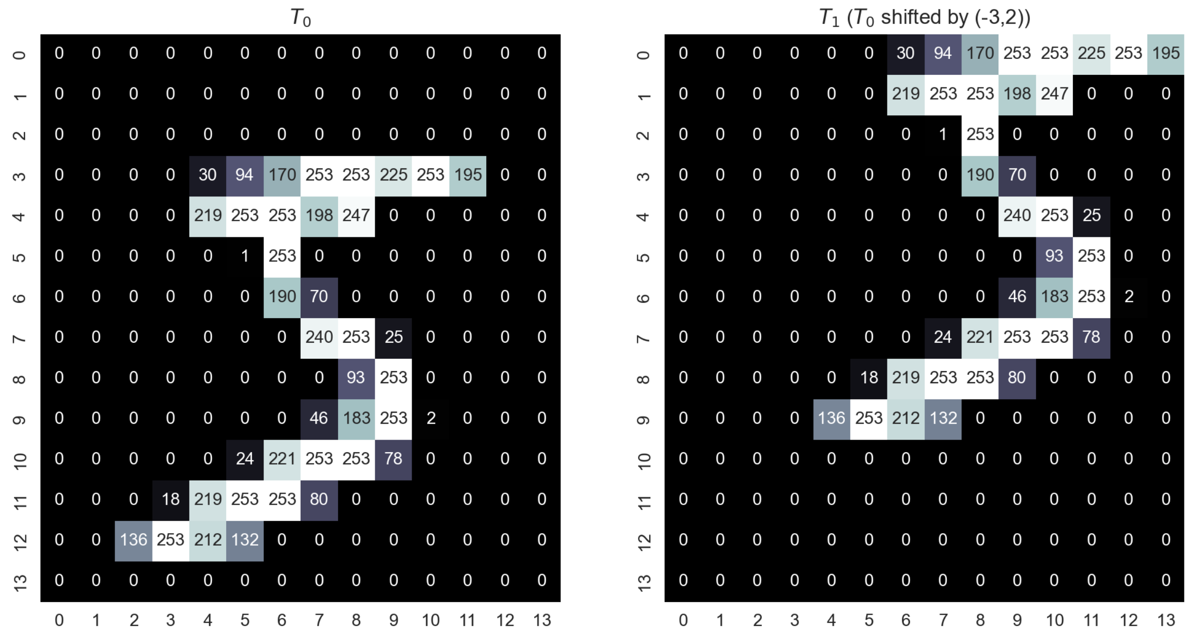
```
fig,ax = plt.subplots(1,2,figsize=(12,5))
ax[0].plot(x,g0,label='Signal $g_0$')
ax[0].plot(x,g1,label='Signal $g_1$')
ax[0].legend()
ax[0].axvline(4,color='red',lw=2)
ax[0].axvline(10,color='lightgreen',lw=2)
ax[0].set_title('Signals')

ax[1].plot(x-10,np.fft.fftshift(cg0g1))
ax[1].axvline(0,color='lightgreen',lw=2)
ax[1].axvline(6,color='red',lw=2);
ax[1].set_title('Cross correlation between $g_0$ and $g_1$');
```



Let's make some test data

We use a 'five' from the MNIST data set of handwritten numbers

```
import numpy as np
import matplotlib.pyplot as plt
from skimage.io import imread
%matplotlib inline
bw_img = np.load('data/five.npy') # A 'five' from the mnist data set

shift_img = np.roll(np.roll(bw_img, -3, axis=0), 2, axis=1)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6), dpi=100)
sns.heatmap(bw_img, ax=ax1, cbar=False, annot=True, fmt='d', cmap='bone'), ax1.set_
↪title('$T_0$')
sns.heatmap(shift_img, ax=ax2, cbar=False, annot=True, fmt='d', cmap='bone') , ax2.
↪set_title('$T_1$ ($T_0$ shifted by (-3,2))');
```



Demonstration of the correlation in space

```
from matplotlib.animation import FuncAnimation
from IPython.display import HTML
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5),dpi=200)

mx, my = np.meshgrid(np.arange(-4, 6, 2),
                     np.arange(-4, 6, 2))

nx = mx.ravel()
ny = my.ravel()
out_score = np.zeros(nx.shape, dtype=np.float32)


def update_frame(i):
    a_img = bw_img
    b_img = np.roll(np.roll(shift_img, nx[i], axis=1), ny[i], axis=0)
    ax1.cla()
    sns.heatmap(a_img, ax=ax1, cbar=False, annot=True, fmt='0.0f', cmap='bone')
    ax2.cla()
    sns.heatmap(b_img, ax=ax2, cbar=False, annot=True, fmt='0.0f', cmap='bone')

    out_score[i] = np.mean(a_img*b_img)
    ax3.cla()
    sns.heatmap(out_score.reshape(mx.shape), ax=ax3,
                cbar=False, annot=True, fmt='2.1f', cmap='viridis')
    ax3.set_xticklabels(mx[0, :])
    ax3.set_yticklabels(my[:, 0])
    ax1.set_title('Iteration #{}'.format(i+1))
    ax2.set_title('X-Offset: {0}\nY-Offset: {1}'.format(nx[i], ny[i]))
    ax3.set_title(r'$\langle I_0(\vec{x}) I_1(\vec{x}+\vec{r}) \rangle$')
```

(continues on next page)

(continued from previous page)

```
# write animation frames
anim_code = FuncAnimation(fig,
                          update_frame,
                          frames=len(nx),
                          interval=300,
                          repeat_delay=4000)

anim_code.save('movies/spacecorrelation_animation.mp4', fps=5, extra_args=['-vcodec',
↪'libx264'])
plt.close('all')
```

0.16.9 Other metrics

Mean Squared Error

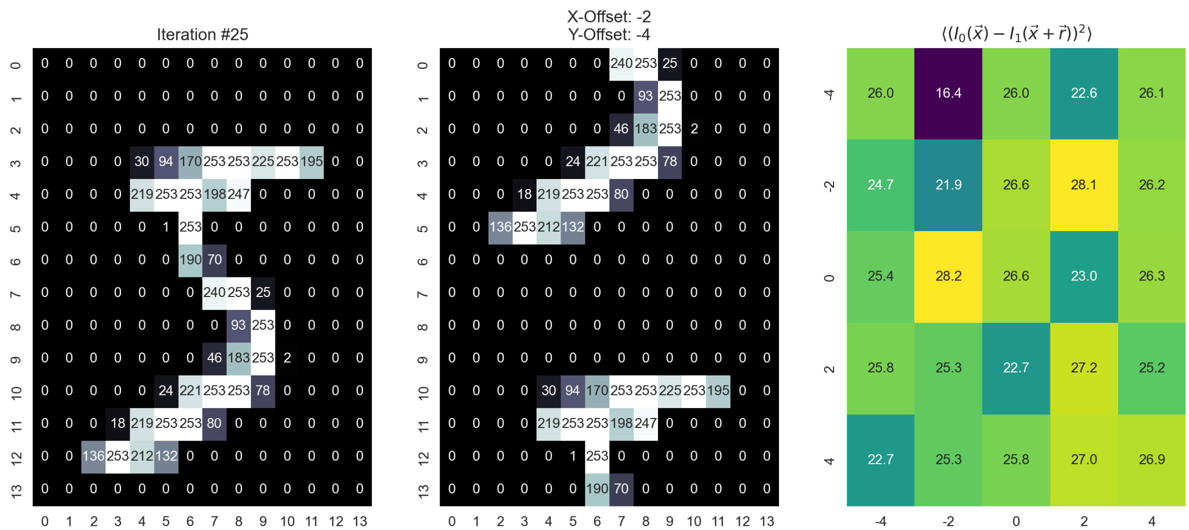
We can also use

- MSE
- or RMSE

and look for minima

```
out_score = np.zeros(nx.shape, dtype=np.float32)

for i in range(len(nx)):
    a_img = bw_img
    b_img = np.roll(np.roll(shift_img, nx[i], axis=1), ny[i], axis=0)
    out_score[i] = np.mean(np.square(a_img-b_img))


# get the minimum
i_min = np.argmin(out_score)
b_img = np.roll(np.roll(shift_img, nx[i_min], axis=1), ny[i_min], axis=0)
```

```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 6))
sns.heatmap(a_img, ax=ax1, cbar=False, annot=True, fmt='d', cmap='bone')
sns.heatmap(b_img, ax=ax2, cbar=False, annot=True, fmt='d', cmap='bone')
sns.heatmap(out_score.reshape(mx.shape), ax=ax3, cbar=False,
            annot=True, fmt='2.1f', cmap='viridis')

ax3.set_xticklabels(mx[0, :])
ax3.set_yticklabels(my[:, 0])
ax1.set_title('Iteration #{}'.format(i+1))
ax2.set_title('X-Offset: %d\nY-Offset: %d' % (nx[i_min], ny[i_min]))
ax3.set_title(r'$\langle (I_0(\vec{x})-I_1(\vec{x}+\vec{r}))^2 \rangle$');
```

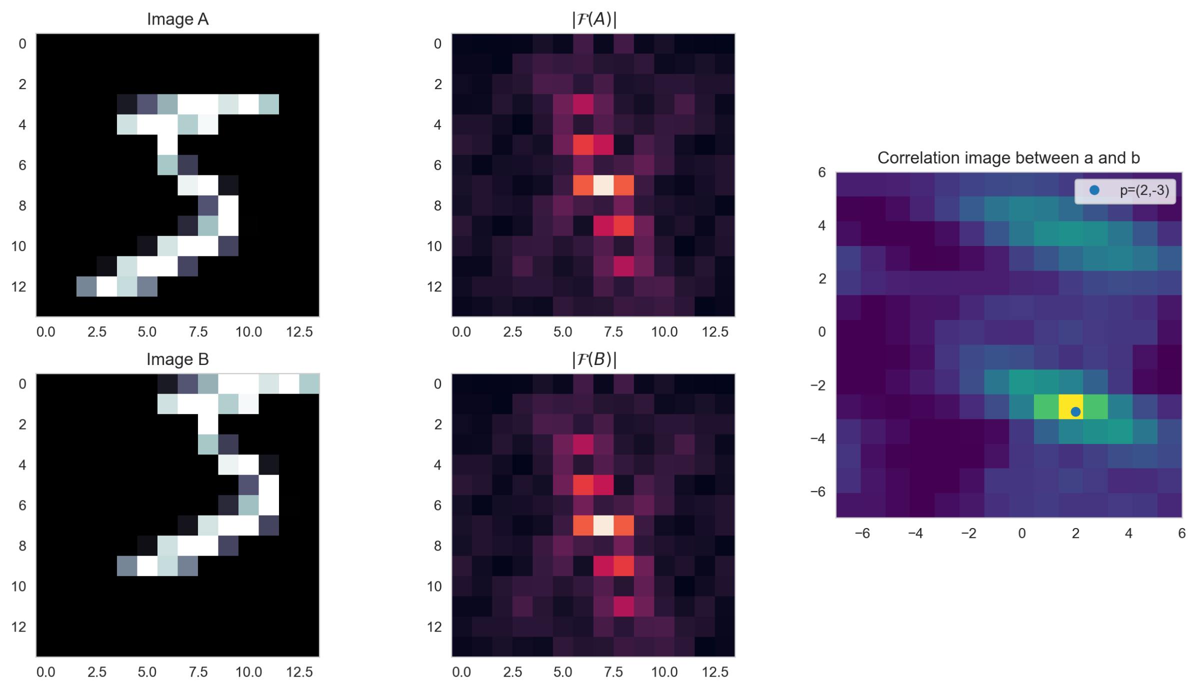


0.16.10 Correlation using the Fourier transform

$$C_{I_0,I_1}(\vec{r}) = \mathcal{F}^{-1}\{\mathcal{F}\{I_0\} \cdot \mathcal{F}\{I_1\}^*\}$$

```
plt.figure(figsize=(15,8))
plt.subplot(2,3,1); plt.imshow(bw_img,cmap='bone'); plt.title('Image A')
plt.subplot(2,3,4); plt.imshow(shift_img,cmap='bone'); plt.title('Image B')

fa=(np.fft.fft2(bw_img));
fb=(np.fft.fft2(shift_img));

plt.subplot(2,3,2); plt.imshow(np.abs(np.fft.fftshift(fa))); plt.title('$|\mathcal{F}
↪(A)|$')
plt.subplot(2,3,5); plt.imshow(np.abs(np.fft.fftshift(fb))); plt.title('$|\mathcal{F}
↪(B)|$')

f=fb*np.conjugate(fa);
co=np.abs(np.fft.fftshift(np.fft.ifft2(f)));
plt.subplot(1,3,3)
plt.imshow(np.abs(co), extent = [-7 , 6, -7 , 6], cmap='viridis',origin='lower');
plt.title('Correlation image between a and b');
plt.plot(2,-3,'o',label='p=(2,-3)')
plt.legend()
```

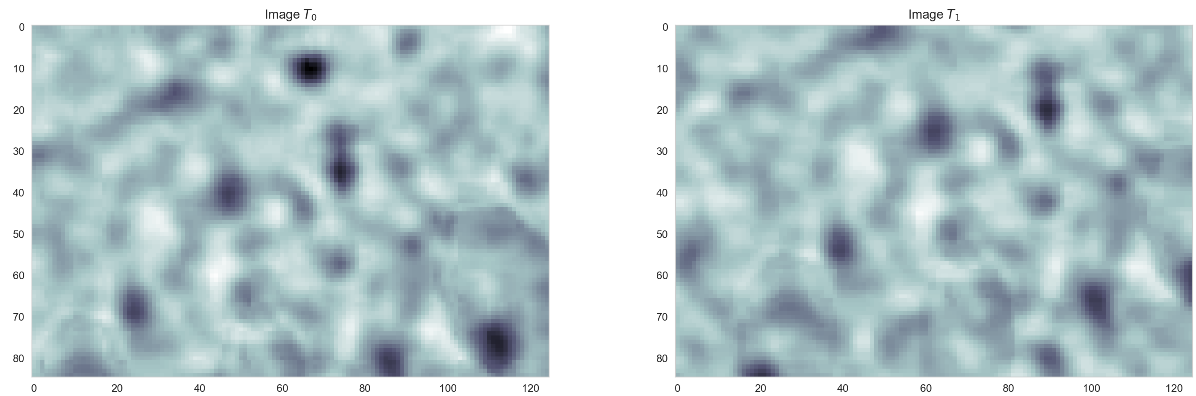
```
<matplotlib.legend.Legend at 0x145614d30>
```



0.16.11 Real Example

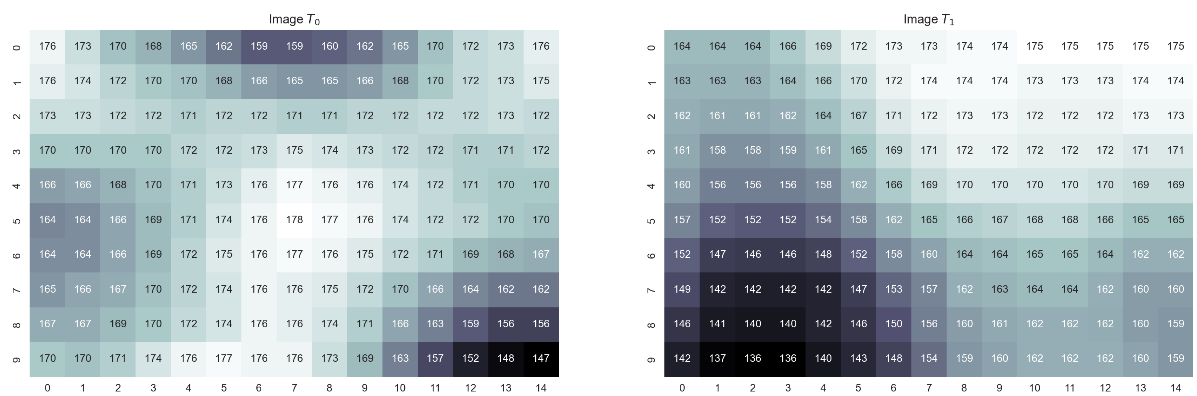
Bone Slice Registration

```
import numpy as np
from skimage.filters import median
import seaborn as sns
import matplotlib.pyplot as plt
from skimage.io import imread
%matplotlib inline
full_img = imread("figures/bonegfiltslice.png").mean(axis=2)
full_shift_img = median(
    np.roll(np.roll(full_img, -15, axis=0), 15, axis=1), np.ones((1, 3)))

def g_roi(x): return x[5:90, 150:275]

bw_img = g_roi(full_img)

shift_img = g_roi(full_shift_img)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 6), dpi=100)
ax1.imshow(bw_img, cmap='bone'),ax1.set_title('Image $T_0$')
ax2.imshow(shift_img, cmap='bone',vmin=bw_img.min(),vmax=bw_img.max()),ax2.set_title(
↪'Image $T_1$');
```



0.16.12 Let's look at a smaller region

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 6), dpi=100)

def g_roi(x): return x[20:30, 210:225]

sns.heatmap(g_roi(full_img), ax=ax1, cbar=False,
            annot=True, fmt='0.0f', cmap='bone'), ax1.set_title('Image $T_0$')
sns.heatmap(g_roi(full_shift_img), ax=ax2, cbar=False,
            annot=True, fmt='0.0f', cmap='bone'); ax2.set_title('Image $T_1$');
```




```
from matplotlib.animation import FuncAnimation
from IPython.display import HTML
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15,5),dpi=200)

def g_roi(x): return x[20:30:2, 210:225:2]

mx, my = np.meshgrid(np.arange(-10, 12, 4),
                     np.arange(-10, 12, 4))

nx = mx.ravel()
ny = my.ravel()
out_score = np.zeros(nx.shape, dtype=np.float32)

def update_frame(i):
    a_img = g_roi(full_img)
```

(continues on next page)

(continued from previous page)

```
    b_img = g_roi(np.roll(np.roll(full_shift_img, nx[i], axis=1), ny[i], axis=0))
    ax1.cla(), sns.heatmap(a_img, ax=ax1, cbar=False, annot=True, fmt='0.0f', cmap=
↪'bone')
    ax2.cla(), sns.heatmap(b_img, ax=ax2, cbar=False, annot=True, fmt='0.0f', cmap=
↪'bone')
    out_score[i] = np.mean(np.square(a_img-b_img))
    ax3.cla(), sns.heatmap(out_score.reshape(mx.shape), ax=ax3, cbar=False,␣
↪annot=True, fmt='2.1f', cmap='RdBu')
    ax1.set_title('Iteration #{}'.format(i+1))
    ax2.set_title('X-Offset: %d\nY-Offset: %d' % (2*nx[i], 2*ny[i]))
    ax3.set_xticklabels(mx[0, :])
    ax3.set_yticklabels(my[:, 0])
    ax3.set_title(r'$\langle (I_0(\vec{x})-I_1(\vec{x}+\vec{r}))^2 \rangle$');

# write animation frames
anim_code = FuncAnimation(fig,
                          update_frame,
                          frames=len(nx),
                          interval=300,
                          repeat_delay=2000)
anim_code.save('movies/spacecorrelation_bone.mp4', fps=5, extra_args=['-vcodec',
↪'libx264'])
plt.close('all')
```

Checking the results

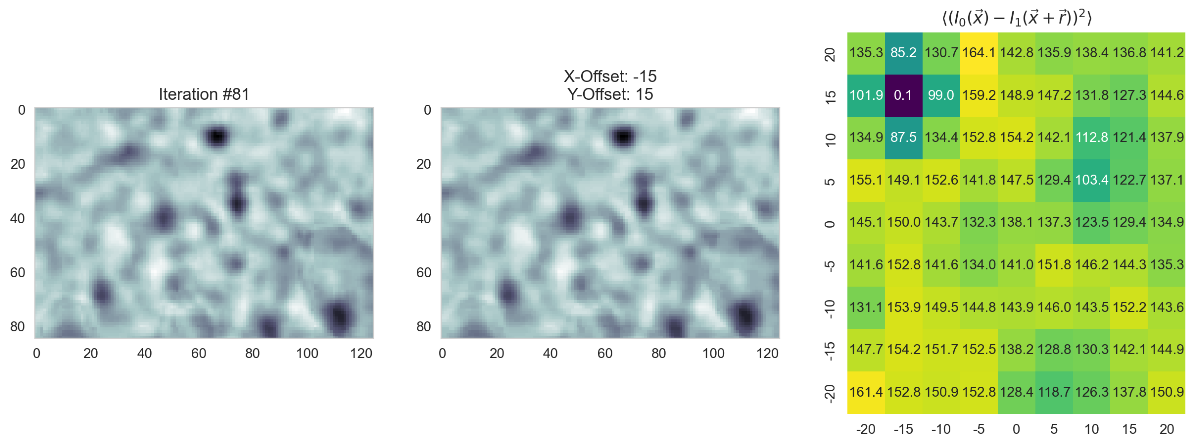
The analysis resulted in a minimum at displacement $\Delta x = -15$, $\Delta y = 15$

```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5), dpi=100)


mx, my = np.meshgrid(np.arange(-20, 25, 5),
                     np.arange(-20, 25, 5))

nx = mx.ravel()
ny = my.ravel()
out_score = np.zeros(nx.shape, dtype=np.float32)

out_score = np.zeros(nx.shape, dtype=np.float32)


def g_roi(x): return x[5:90, 150:275]


for i in range(len(nx)):
    a_img = g_roi(full_img)
    b_img = g_roi(
        np.roll(np.roll(full_shift_img, nx[i], axis=1), ny[i], axis=0))
    out_score[i] = np.mean(np.square(a_img-b_img))

# get the minimum
i_min = np.argmin(out_score)
b_img = g_roi(np.roll(np.roll(full_shift_img, nx[i_min], axis=1), ny[i_min], axis=0))
```

(continues on next page)

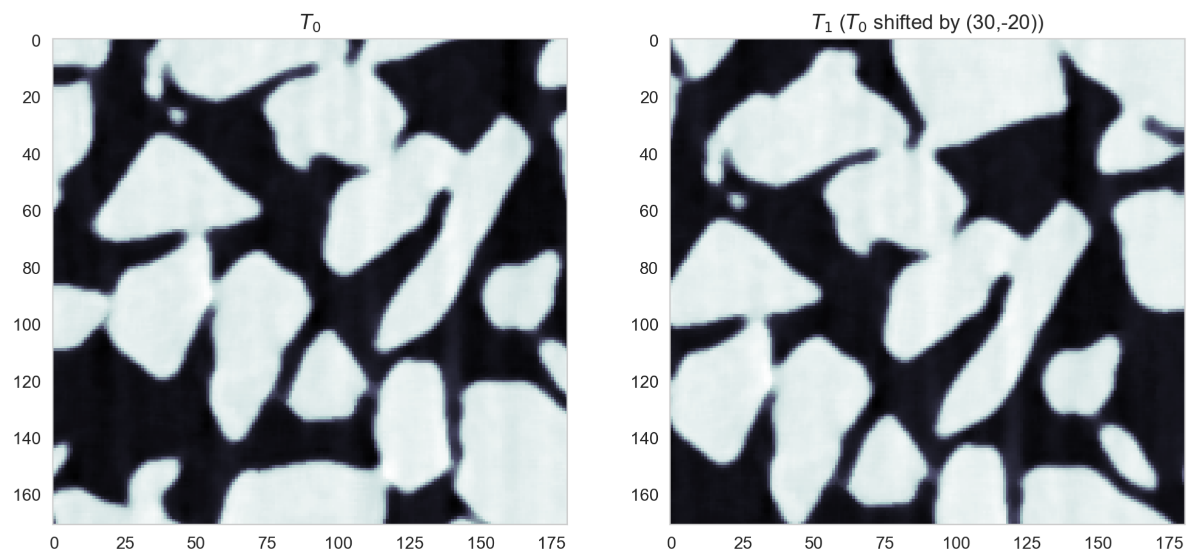
(continued from previous page)

```
ax1.imshow(a_img, cmap='bone'), ax1.set_title('$T_0$')
ax2.imshow(b_img, cmap='bone'), ax2.set_title('$T_1$ Registered')
sns.heatmap(out_score.reshape(mx.shape), ax=ax3, cbar=False,annot=True, fmt='2.1f',␣
↪cmap='viridis')
ax3.invert_yaxis()
ax3.set_xticklabels(mx[0, :]), ax3.set_yticklabels(my[:, 0])
ax1.set_title('Iteration #{}'.format(i+1))
ax2.set_title('X-Offset: %d\nY-Offset: %d' % (nx[i_min], ny[i_min]))
ax3.set_title(r'$\langle (I_0(\vec{x})-I_1(\vec{x}+\vec{r}))^2 \rangle$');
```



Sand grains

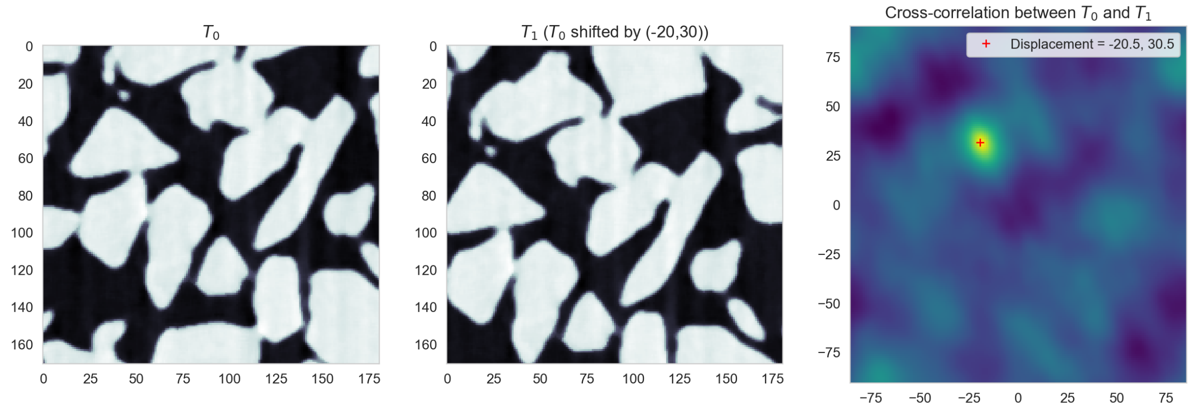
```
bw_sand = plt.imread('data/sand.png').astype(float) # Some sand grains

shift_sand = bw_sand[:-30,20:]
bw_sand = bw_sand[30:,:shift_sand.shape[1]]

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6), dpi=100)
ax1.imshow(bw_sand,cmap='bone'), ax1.set_title('$T_0$')
ax2.imshow(shift_sand, cmap='bone') , ax2.set_title('$T_1$ ($T_0$ shifted by (30,-20))
↪');
```



Correlation in Fourier space

```
fg0=np.fft.fft2(bw_sand)
fg1=np.fft.fft2(shift_sand)
fg0g1=fg1*np.conj(fg0)
cg0g1=np.real(np.fft.fftshift(np.fft.ifft2(fg0g1)))

pos=np.unravel_index(np.argmax(cg0g1[::-1]),cg0g1.shape)
x=pos[1]-cg0g1.shape[1]/2
y=-(pos[0]-cg0g1.shape[0]/2)
```

```
fig,ax=plt.subplots(1,3,figsize=(15,5))
ax[0].imshow(bw_sand,cmap='bone'), ax[0].set_title('$T_0$')
ax[1].imshow(shift_sand, cmap='bone') , ax[1].set_title('$T_1$ ($T_0$ shifted by (-20,
↪30))');

ax[2].imshow(cg0g1[::-1],cmap='viridis',extent=[-cg0g1.shape[0]/2,cg0g1.shape[0]/2,-
↪cg0g1.shape[1]/2,cg0g1.shape[1]/2])
ax[2].plot(x+1,y+1,'r+',label='Displacement = {0}, {1}'.format(x,y));
ax[2].legend();
ax[2].set_title('Cross-correlation between $T_0$ and $T_1$');
```



0.17 Registration

Before any meaningful tracking tasks can be performed, the first step is to register the measurements so they are all on the same coordinate system.

Often the registration can be done along with the tracking by separating the movement into actual sample movement and other (camera, setup, etc) if the motion of either the sample or the other components can be well modeled.

In medicine this is frequently needed because different scanners produce different kinds of outputs with different scales, positioning and resolutions. This is also useful for 'follow-up' scans with patients to identify how a disease has progressed. With scans like chest X-rays it isn't uncommon to have multiple (some patients have hundreds) all taken under different conditions

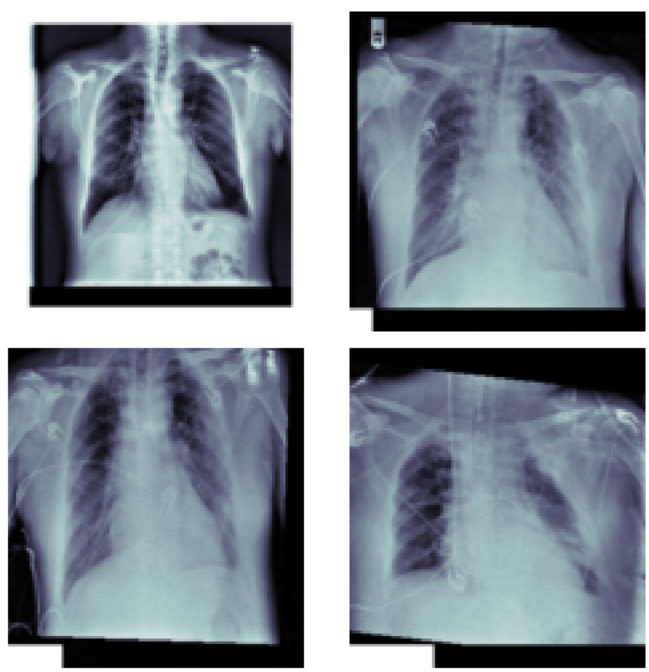


Fig. 1: A series of chest X-ray images taken at different times.

0.17.1 The Process

We informally followed a process before when trying to match the two images together, but we want to make this more generic for a larger spectrum of problems.

We thus follow the model set forward by tools like ITK with the components divided into the input data:

- *Moving Image*
- and *Fixed Image* sometimes called *Reference Image*).

The algorithmic components

- The *Transform* operation to transform the moving image.
- The *interpolator* to handle bringing all of the points onto a pixel grid.
- The *Metric* which is the measure of how well the transformed moving image and fixed image match
- and finally the *Optimizer* that tries to find the best solution

```
from IPython.display import SVG
from subprocess import check_output
import pydot
import os


def show_graph(graph):
    try:
        return SVG(graph.create_svg())
    except AttributeError as e:
        output = check_output('dot -Tsvg', shell=True,
                              input=g.to_string().encode())
        return SVG(output.decode())


g = pydot.Graph(graph_type='digraph')
fixed_img = pydot.Node('Fixed Image\nReference Image',
                       shape='folder', style="filled", fillcolor="lightgreen")
moving_img = pydot.Node('Moving Image', shape='folder',
                        style="filled", fillcolor="lightgreen")
trans_obj = pydot.Node('Transform', shape='box',
                       style='filled', fillcolor='yellow')
g.add_node(fixed_img)
g.add_node(moving_img)
g.add_node(trans_obj)
g.add_edge(pydot.Edge(fixed_img, 'Metric'))
g.add_edge(pydot.Edge(moving_img, 'Interpolator'))
g.add_edge(pydot.Edge(trans_obj, 'Interpolator', label='Transform Parameters'))
g.add_edge(pydot.Edge('Interpolator', 'Metric'))
show_graph(g)
```

```
<IPython.core.display.SVG object>
```

```
g.add_edge(pydot.Edge('Metric', 'Optimizer'))
g.add_edge(pydot.Edge('Optimizer', trans_obj))
show_graph(g)
```

```
<IPython.core.display.SVG object>
```

0.17.2 Images

Fixed Image

The fixed image (or reference image) is the image that will be left untouched and used for comparison

Moving Image

The moving image will be transformed (translated, scaled, rotated, deformed, …) to try and match as closely as possible the fixed image.

0.17.3 Transform

The transform specifies the transformations which can take place on the moving image, a number of different types are possible, but the most frequent types are listed below.

- Affine
- Translation
- Scaling
- Deformable
- Shearing

0.17.4 Interpolator

The interpolator is the component applies the transform to the moving image. The common ways of interpolating are

- Nearest Neighbor
- Bilinear
- Bicubic
- Bspline
- …

0.17.5 Metric

The metric is how the success of the matching of the two images is measured. The goal is to measure similarity between images.

- Mean Squared Error - the simplist metric to use just recording the raw difference, but often this can lead to unusual matches since noise and uneven illumination can lead to high MSE for images that match well.
- SSIM similarity metric
- Correlation Factor

0.17.6 Optimizer

The optimizer component is responsible for updating the parameters based on the metric. A standard approach with this is gradient descent where the gradient is calculated and a small step (determined by the learning rate) is taken in the direction of maximum descent.

- Gradient Descent
- Adam
- Stochastic Gradient Descent
- AdaGrad
- AdaDelta

0.17.7 Our tracker

```
from IPython.display import Image,SVG

g = pydot.Dot(graph_type='digraph')
fixed_img = pydot.Node('Fixed Image\nReference Image',
                       shape='folder', style="filled", fillcolor="lightgreen")
moving_img = pydot.Node('Moving Image', shape='folder',
                        style="filled", fillcolor="lightgreen")
trans_obj = pydot.Node('Transform', shape='box',
                       style='filled', fillcolor='yellow')
g.add_node(fixed_img)
g.add_node(moving_img)
g.add_node(trans_obj)
g.add_edge(pydot.Edge(fixed_img, 'Metric\nMean Squared Error'))
g.add_edge(pydot.Edge(moving_img, 'Interpolator\nNearest Neighbor'))
g.add_edge(pydot.Edge(trans_obj, 'Interpolator\nNearest Neighbor',
                      label='Transform Parameters'))
g.add_edge(pydot.Edge('Interpolator\nNearest Neighbor',
                      'Metric\nMean Squared Error'))
#g.add_edge(pydot.Edge('Metric\nMean Squared Error', 'Optimizer\nGrid Search', style
↪= ''))
g.add_edge(pydot.Edge('Optimizer\nGrid Search', trans_obj))
show_graph(g)
```

```
<IPython.core.display.SVG object>
```

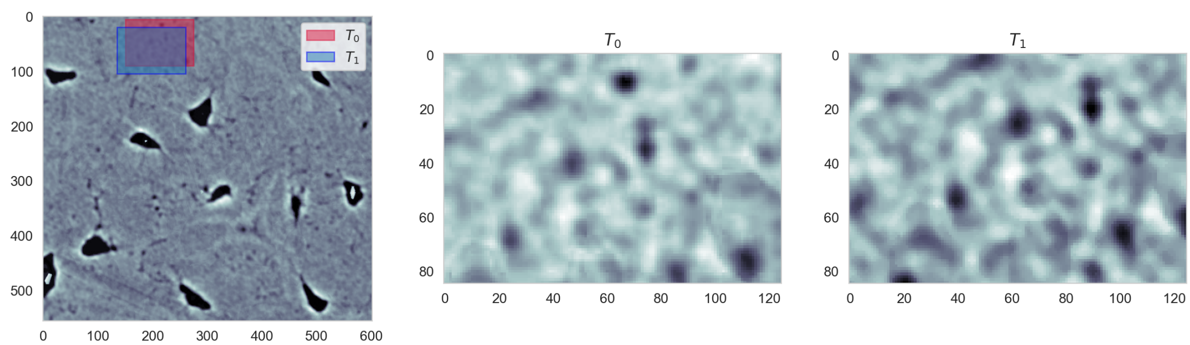
0.17.8 Registration of the bone image

```
import numpy as np
from skimage.filters import median
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
from skimage.io import imread
%matplotlib inline
full_img = imread("figures/bonegfiltslice.png")[:,:,0:3].mean(axis=2)
full_shift_img = median(
```

(continues on next page)

(continued from previous page)

```
    np.roll(np.roll(full_img, -15, axis=0), 15, axis=1), footprint=np.ones((3, 3)))

def g_roi(x): return x[5:90, 150:275]

bw_img = g_roi(full_img)
shift_img = g_roi(full_shift_img)

fig, (ax0,ax1, ax2) = plt.subplots(1, 3, figsize=(15, 4))
ax0.imshow(full_img,cmap='bone')
r=Rectangle((150,5),125,85,ec='crimson',fc='crimson',alpha=0.5,label=r"$T_0$")
ax0.add_patch(r)
r=Rectangle((135,20),125,85,ec='blue',alpha=0.5,label=r"$T_1$")
ax0.add_patch(r)
ax0.legend()
ax1.imshow(bw_img, cmap='bone')
ax1.set_title('$T_0$')
ax2.imshow(shift_img, cmap='bone')
ax2.set_title('$T_1$');
```




```
%%file affine_op.py
import tensorflow as tf

"""
Code taken from https://github.com/kevinzakka/spatial-transformer-network/blob/master/
↪transformer.py
"""


def affine_transform(input_fmap, theta, out_dims=None, **kwargs):
    """
    Spatial Transformer Network layer implementation as described in [1].
    The layer is composed of 3 elements:
    - localisation_net: takes the original image as input and outputs
      the parameters of the affine transformation that should be applied
      to the input image.
    - affine_grid_generator: generates a grid of (x,y) coordinates that
      correspond to a set of points where the input should be sampled
      to produce the transformed output.
    - bilinear_sampler: takes as input the original image and the grid
      and produces the output image using bilinear interpolation.
    Input
    -----
    - input_fmap: output of the previous layer. Can be input if spatial
```

(continues on next page)

(continued from previous page)

```
      transformer layer is at the beginning of architecture. Should be
      a tensor of shape (B, H, W, C).
    - theta: affine transform tensor of shape (B, 6). Permits cropping,
      translation and isotropic scaling. Initialize to identity matrix.
      It is the output of the localization network.
    Returns
    -------
    - out_fmap: transformed input feature map. Tensor of size (B, H, W, C).
    Notes
    -----
    [1]: 'Spatial Transformer Networks', Jaderberg et. al,
         (https://arxiv.org/abs/1506.02025)
    """
    # grab input dimensions
    B = tf.shape(input_fmap)[0]
    H = tf.shape(input_fmap)[1]
    W = tf.shape(input_fmap)[2]
    C = tf.shape(input_fmap)[3]

    # reshape theta to (B, 2, 3)
    theta = tf.reshape(theta, [B, 2, 3])

    # generate grids of same size or upsample/downsample if specified
    if out_dims:
        out_H = out_dims[0]
        out_W = out_dims[1]
        batch_grids = affine_grid_generator(out_H, out_W, theta)
    else:
        batch_grids = affine_grid_generator(H, W, theta)

    x_s = batch_grids[:, 0, :, :]
    y_s = batch_grids[:, 1, :, :]

    # sample input with grid to get output
    out_fmap = bilinear_sampler(input_fmap, x_s, y_s)

    return out_fmap


def get_pixel_value(img, x, y):
    """
    Utility function to get pixel value for coordinate
    vectors x and y from a  4D tensor image.
    Input
    -----
    - img: tensor of shape (B, H, W, C)
    - x: flattened tensor of shape (B*H*W, )
    - y: flattened tensor of shape (B*H*W, )
    Returns
    -------
    - output: tensor of shape (B, H, W, C)
    """
    shape = tf.shape(x)
    batch_size = shape[0]
    height = shape[1]
    width = shape[2]
```

(continues on next page)

(continued from previous page)

```
    batch_idx = tf.range(0, batch_size)
    batch_idx = tf.reshape(batch_idx, (batch_size, 1, 1))
    b = tf.tile(batch_idx, (1, height, width))

    indices = tf.stack([b, y, x], 3)

    return tf.gather_nd(img, indices)


def affine_grid_generator(height, width, theta):
    """
    This function returns a sampling grid, which when
    used with the bilinear sampler on the input feature
    map, will create an output feature map that is an
    affine transformation [1] of the input feature map.
    Input
    -----
    - height: desired height of grid/output. Used
      to downsample or upsample.
    - width: desired width of grid/output. Used
      to downsample or upsample.
    - theta: affine transform matrices of shape (num_batch, 2, 3).
      For each image in the batch, we have 6 theta parameters of
      the form (2x3) that define the affine transformation T.
    Returns
    -------
    - normalized gird (-1, 1) of shape (num_batch, 2, H, W).
      The 2nd dimension has 2 components: (x, y) which are the
      sampling points of the original image for each point in the
      target image.
    Note
    ----
    [1]: the affine transformation allows cropping, translation,
         and isotropic scaling.
    """
    # grab batch size
    num_batch = tf.shape(theta)[0]

    # create normalized 2D grid
    x = tf.linspace(-1.0, 1.0, width)
    y = tf.linspace(-1.0, 1.0, height)
    x_t, y_t = tf.meshgrid(x, y)

    # flatten
    x_t_flat = tf.reshape(x_t, [-1])
    y_t_flat = tf.reshape(y_t, [-1])

    # reshape to [x_t, y_t , 1] - (homogeneous form)
    ones = tf.ones_like(x_t_flat)
    sampling_grid = tf.stack([x_t_flat, y_t_flat, ones])

    # repeat grid num_batch times
    sampling_grid = tf.expand_dims(sampling_grid, axis=0)
    sampling_grid = tf.tile(sampling_grid, tf.stack([num_batch, 1, 1]))
```

(continues on next page)

(continued from previous page)

```
    # cast to float32 (required for matmul)
    theta = tf.cast(theta, 'float32')
    sampling_grid = tf.cast(sampling_grid, 'float32')

    # transform the sampling grid - batch multiply
    batch_grids = tf.matmul(theta, sampling_grid)
    # batch grid has shape (num_batch, 2, H*W)

    # reshape to (num_batch, H, W, 2)
    batch_grids = tf.reshape(batch_grids, [num_batch, 2, height, width])

    return batch_grids


def bilinear_sampler(img, x, y):
    """
    Performs bilinear sampling of the input images according to the
    normalized coordinates provided by the sampling grid. Note that
    the sampling is done identically for each channel of the input.
    To test if the function works properly, output image should be
    identical to input image when theta is initialized to identity
    transform.
    Input
    -----
    - img: batch of images in (B, H, W, C) layout.
    - grid: x, y which is the output of affine_grid_generator.
    Returns
    -------
    - interpolated images according to grids. Same size as grid.
    """
    # prepare useful params
    B = tf.shape(img)[0]
    H = tf.shape(img)[1]
    W = tf.shape(img)[2]
    C = tf.shape(img)[3]

    max_y = tf.cast(H - 1, 'int32')
    max_x = tf.cast(W - 1, 'int32')
    zero = tf.zeros([], dtype='int32')

    # cast indices as float32 (for rescaling)
    x = tf.cast(x, 'float32')
    y = tf.cast(y, 'float32')

    # rescale x and y to [0, W/H]
    x = 0.5 * ((x + 1.0) * tf.cast(W, 'float32'))
    y = 0.5 * ((y + 1.0) * tf.cast(H, 'float32'))

    # grab 4 nearest corner points for each (x_i, y_i)
    # i.e. we need a rectangle around the point of interest
    x0 = tf.cast(tf.floor(x), 'int32')
    x1 = x0 + 1
    y0 = tf.cast(tf.floor(y), 'int32')
    y1 = y0 + 1

    # clip to range [0, H/W] to not violate img boundaries
```

(continues on next page)

(continued from previous page)

```
    x0 = tf.clip_by_value(x0, zero, max_x)
    x1 = tf.clip_by_value(x1, zero, max_x)
    y0 = tf.clip_by_value(y0, zero, max_y)
    y1 = tf.clip_by_value(y1, zero, max_y)

    # get pixel value at corner coords
    Ia = get_pixel_value(img, x0, y0)
    Ib = get_pixel_value(img, x0, y1)
    Ic = get_pixel_value(img, x1, y0)
    Id = get_pixel_value(img, x1, y1)

    # recast as float for delta calculation
    x0 = tf.cast(x0, 'float32')
    x1 = tf.cast(x1, 'float32')
    y0 = tf.cast(y0, 'float32')
    y1 = tf.cast(y1, 'float32')

    # calculate deltas
    wa = (x1-x) * (y1-y)
    wb = (x1-x) * (y-y0)
    wc = (x-x0) * (y1-y)
    wd = (x-x0) * (y-y0)

    # add dimension for addition
    wa = tf.expand_dims(wa, axis=3)
    wb = tf.expand_dims(wb, axis=3)
    wc = tf.expand_dims(wc, axis=3)
    wd = tf.expand_dims(wd, axis=3)

    # compute output
    out = tf.add_n([wa*Ia, wb*Ib, wc*Ic, wd*Id])

    return out
```

```
Overwriting affine_op.py
```

```
#import tensorflow as tf
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
from affine_op import affine_transform
g = tf.Graph()

with g.as_default():
    init = tf.global_variables_initializer()
    # tf Graph Input
    fixed_img = tf.placeholder("float", shape=(
        1, None, None, 1), name='FixedImage')
    moving_img = tf.placeholder("float", shape=(
        1, None, None, 1), name='MovingImage')
    # Initialize the variables (i.e. assign their default value)

    with tf.name_scope('transform_parameters'):  # Set transform parameters
        x_offset = tf.Variable(0.0, name="x_offset")
        y_offset = tf.Variable(0.0, name="y_offset")
        # we keep scale and rotation fixed
```

(continues on next page)

(continued from previous page)

```
        scale = tf.placeholder("float", shape=tuple(), name="scale")
        rotation = tf.placeholder("float", shape=tuple(), name="rotation")

    with tf.name_scope('transformer_and_interpolator'):
        flat_mat = tf.tile([tf.cos(rotation), -tf.sin(rotation), x_offset,
                            tf.sin(rotation), tf.cos(rotation), y_offset], (1,))
        flat_mat = tf.reshape(flat_mat, (1, 6))
        trans_tensor = affine_transform(moving_img, flat_mat)

    with tf.name_scope('metric'):
        mse = tf.reduce_mean(
            tf.square(fixed_img-trans_tensor), name='MeanSquareError')
        optimizer = tf.train.GradientDescentOptimizer(1e-5).minimize(mse)
```

```
WARNING:tensorflow:From /Users/kaestner/miniconda3/lib/python3.9/site-packages/
↪tensorflow/python/compat/v2_compat.py:107: disable_resource_variables (from␣
↪tensorflow.python.ops.variable_scope) is deprecated and will be removed in a␣
↪future version.
Instructions for updating:
non-resource variables are not supported in the long term
```

```
import numpy as np
from IPython.display import clear_output, Image, display, HTML

def strip_consts(graph_def, max_const_size=32):
    """Strip large constant values from graph_def."""
    strip_def = tf.GraphDef()
    for n0 in graph_def.node:
        n = strip_def.node.add()
        n.MergeFrom(n0)
        if n.op == 'Const':
            tensor = n.attr['value'].tensor
            size = len(tensor.tensor_content)
            if size > max_const_size:
                tensor.tensor_content = "<stripped %d bytes>" % size
    return strip_def


def show_graph(graph_def, max_const_size=32):
    """Visualize TensorFlow graph."""
    if hasattr(graph_def, 'as_graph_def'):
        graph_def = graph_def.as_graph_def()
    strip_def = strip_consts(graph_def, max_const_size=max_const_size)
    code = """
        <script src="//cdnjs.cloudflare.com/ajax/libs/polymer/0.3.3/platform.js"></
↪script>
        <script>
          function load() {{
            document.getElementById("{id}").pbtxt = {data};
          }}
        </script>
        <link rel="import" href="https://tensorboard.appspot.com/tf-graph-basic.build.
↪html" onload=load()>
        <div style="height:600px">
          <tf-graph-basic id="{id}"></tf-graph-basic>
```

(continues on next page)

(continued from previous page)

```
        </div>
    """.format(data=repr(str(strip_def)), id='graph'+str(np.random.rand()))

    iframe = """
        <iframe seamless style="width:1200px;height:620px;border:0" srcdoc="{}"></
↪iframe>
    """.format(code.replace('"', '&quot;'))
    display(HTML(iframe))


show_graph(g)
```

```
<IPython.core.display.HTML object>
```

```
# Start training
from matplotlib.animation import FuncAnimation
from IPython.display import HTML
import numpy as np


def make_feed_dict(f_img, m_img):
    return {fixed_img: np.expand_dims(np.expand_dims(f_img, 0), -1),
            moving_img: np.expand_dims(np.expand_dims(m_img, 0), -1),
            rotation: 0.0}


loss_history = []
optimize_iters = 10
with tf.Session(graph=g) as sess:
    plt.close('all')
    fig, m_axs = plt.subplots(2, 2, figsize=(10, 10), dpi=100)
    #tf.initialize_all_variables().run()
    init = tf.global_variables_initializer()
    # Run the initializer
    sess.run(init)
    # Fit all training data
    const_feed_dict = make_feed_dict(bw_img, shift_img)

    def update_frame(i):
        global loss_history
        (ax1, ax2), (ax4, ax3) = m_axs
        for c_ax in m_axs.flatten():
            c_ax.cla()
            c_ax.axis('off')
        f_mse, x_pos, y_pos, rs_img = sess.run([mse, x_offset, y_offset, trans_
↪tensor],
                                               feed_dict=const_feed_dict)
        loss_history += [f_mse]

        ax1.imshow(bw_img, cmap='bone')
        ax1.set_title('$T_0$')
        ax2.imshow(shift_img, cmap='bone')
        ax2.set_title('$T_1$')
        #ax3.imshow(rs_img[0,:,:,0], cmap = 'bone')
        # ax3.set_title('Output')
```

(continues on next page)

(continued from previous page)

```
        ax4.imshow(bw_img*1.0-rs_img[0, :, :, 0],
                   cmap='RdBu', vmin=-100, vmax=100)
        ax4.set_title('Difference\nMSE: %2.2f' % (f_mse))
        ax3.semilogy(loss_history)
        ax3.set_xlabel('Iteration')
        ax3.set_ylabel('MSE (Log-scale)')
        ax3.axis('on')

        for _ in range(1):
            sess.run(optimizer, feed_dict=const_feed_dict)
    # write animation frames
    anim_code = FuncAnimation(fig,
                              update_frame,
                              frames=optimize_iters,
                              interval=1000,
                              repeat_delay=2000)

    anim_code.save('movies/optimizedregistration_bone1.mp4', fps=5, extra_args=['-
↪vcodec', 'libx264'])
    plt.close('all')
#HTML(anim_code)
```

```
2023-05-11 13:31:57.405483: I tensorflow/compiler/mlir/mlir_graph_optimization_
↪pass.cc:357] MLIR V1 optimization pass is not enabled
2023-05-11 13:31:57.410722: W tensorflow/tsl/platform/profile_utils/cpu_utils.
↪cc:128] Failed to get CPU frequency: 0 Hz
```

```
g_roi = tf.Graph()
with g_roi.as_default():
    init = tf.global_variables_initializer()
    # tf Graph Input
    fixed_img = tf.placeholder("float", shape=(
        1, None, None, 1), name='FixedImage')
    moving_img = tf.placeholder("float", shape=(
        1, None, None, 1), name='MovingImage')
    # Initialize the variables (i.e. assign their default value)

    with tf.name_scope('transform_parameters'):  # Set transform parameters
        x_offset = tf.Variable(0.0, name="x_offset")
        y_offset = tf.Variable(0.0, name="y_offset")
        # we keep rotation fixed
        rotation = tf.placeholder("float", shape=tuple(), name="rotation")

    with tf.name_scope('transformer_and_interpolator'):
        flat_mat = tf.tile([tf.cos(rotation), -tf.sin(rotation), x_offset,
                            tf.sin(rotation), tf.cos(rotation), y_offset], (1,))
        flat_mat = tf.reshape(flat_mat, (1, 6))
        trans_tensor = affine_transform(moving_img, flat_mat)

    with tf.name_scope('metric'):
        diff_tensor = (fixed_img-trans_tensor)[:, 25:75, 25:110, :]
        mse = tf.reduce_mean(tf.square(diff_tensor), name='MeanSquareError')
        optimizer = tf.train.GradientDescentOptimizer(2e-6).minimize(mse)
```

```
# Start training
from matplotlib.animation import FuncAnimation
from IPython.display import HTML
from matplotlib import patches
optimize_iters = 20
loss_history = []
with tf.Session(graph=g_roi) as sess:
    plt.close('all')
    fig, m_axs = plt.subplots(2, 3, figsize=(15, 7), dpi=100)
    init = tf.global_variables_initializer()
    # Run the initializer
    sess.run(init)
    # Fit all training data
    const_feed_dict = make_feed_dict(bw_img, shift_img)

    def update_frame(i):
        global loss_history
        (ax1, ax2, ax5), (ax3, ax4, ax6) = m_axs
        for c_ax in m_axs.flatten():
            c_ax.cla()
            c_ax.axis('off')
        f_mse, x_pos, y_pos, rs_img, diff_img = sess.run([mse, x_offset, y_offset,␣
↪trans_tensor, diff_tensor],
                                                         feed_dict=const_feed_dict)
        loss_history += [f_mse]

        ax1.imshow(bw_img, cmap='bone')
        ax1.set_title('$T_0$')
        ax2.imshow(shift_img, cmap='bone')
        ax2.set_title('$T_1$')
        ax3.imshow(rs_img[0, :, :, 0], cmap='bone')
        ax3.set_title('Output')
        ax4.imshow(bw_img*1.0-rs_img[0, :, :, 0],
                   cmap='RdBu', vmin=-100, vmax=100)
        ax4.set_title('MSE: %2.2f' % (f_mse))
        rect = patches.Rectangle(
            (25, 25), 85, 50, linewidth=2, edgecolor='g', facecolor='none')
        # Add the patch to the Axes
        ax4.add_patch(rect)
        ax5.semilogy(loss_history)
        ax5.set_xlabel('Iteration')
        ax5.set_ylabel('MSE (Log-scale)')
        ax5.axis('on')

        ax6.imshow(diff_img[0, :, :, 0], cmap='RdBu', vmin=-100, vmax=100)
        ax6.set_title('ROI')
        for _ in range(5):
            sess.run(optimizer, feed_dict=const_feed_dict)
    # write animation frames
    anim_code = FuncAnimation(fig,
                              update_frame,
                              frames=optimize_iters,
                              interval=1000,
                              repeat_delay=2000)#.to_html5_video()

    anim_code.save('movies/optimizedregistration_bone.mp4', fps=5, extra_args=['-
↪vcodec', 'libx264'])
```

(continues on next page)

(continued from previous page)

```
    plt.close('all')
#HTML("<video controls loop src='movies/optimizedregistration_bone.mp4' height='500px
↪' type='video/mp4'></video>")
```

Registration with least squared cost function

This code does unfortunately not work…

The original code was implemented using Tensorflow 1 and is deprecated. This is an attempt to implement the same with TF2. It does unfortunately not work yet…

```
def affine_transform(fixed_img, moving_img, transform_params):
    # Define affine transform matrix
    theta = transform_params[0]
    tx    = transform_params[1]
    ty    = transform_params[2]

    transformed=apply_affine_transform(moving_image,theta=theta,tx=tx,ty=ty)

    return transformed


def similarity_loss(fixed_img, moving_img, transform_params):
    # Apply affine transform to moving image
    transformed_img = affine_transform(fixed_img, moving_img, transform_params)

    # Compute mean squared error between fixed and transformed moving images
    mse = tf.reduce_mean(tf.square(fixed_img - transformed_img))

    return mse

# Define the transform parameters as trainable variables
theta = tf.Variable(0.0, dtype=tf.float32, trainable=True)
tx    = tf.Variable(0.0, dtype=tf.float32, trainable=True)
ty    = tf.Variable(0.0, dtype=tf.float32, trainable=True)
transform_params = [theta, tx, ty]

# Define the optimizer
optimizer = tf.optimizers.Adam()

# Define the training loop
for i in range(1000):
    with tf.GradientTape() as tape:
        tape.watch(transform_params)
        loss = similarity_loss(fixed_image, moving_image, transform_params)

    tf.print(transform_params)
    gradients = tape.gradient(loss, transform_params)

    print(gradients)
    optimizer.apply_gradients(zip(gradients, transform_params))

    if i % 100 == 0:
        print(f'Loss at step {i}: {loss.numpy()}')
```

(continues on next page)

(continued from previous page)

```
# Apply the learned transform to the moving image
registered_image = affine_transform(fixed_image, moving_image, transform_params)
```

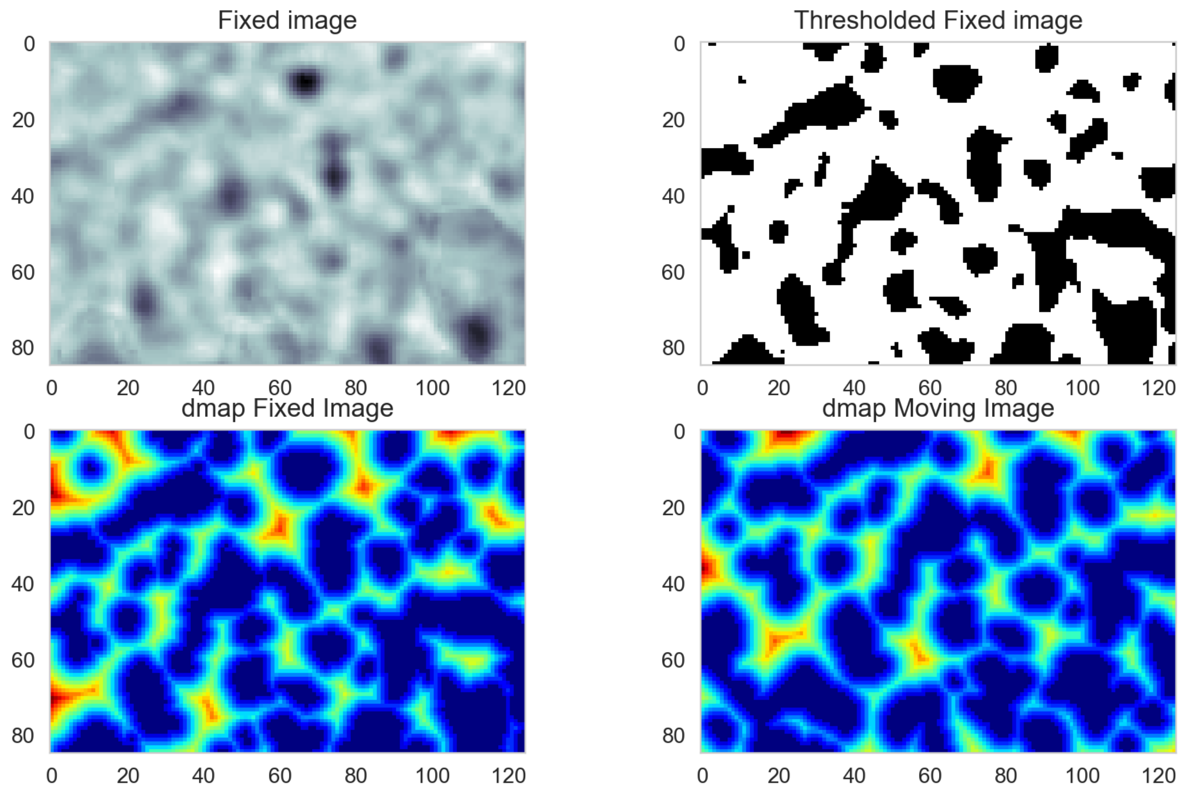
```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[57], line 28
     25 transform_params = [theta, tx, ty]
     27 # Define the optimizer
---> 28 optimizer = tf.optimizers.Adam()
     30 # Define the training loop
     31 for i in range(1000):

AttributeError: module 'tensorflow.compat.v1' has no attribute 'optimizers'
```

0.17.9 Smoother Gradient

We can use a distance map of the segmentation to give us a smoother gradient

```
thresh_img = bw_img > threshold_otsu(bw_img)
dist_start_img = distance_transform_edt(thresh_img)
dist_shift_img = distance_transform_edt(shift_img > threshold_otsu(bw_img))
```

```
fig, [(ax1, ax2), (ax3, ax4)] = plt.subplots(2, 2, figsize=(10,6))
ax1.imshow(bw_img, cmap='bone')
ax1.set_title('Fixed image')
ax2.imshow(thresh_img, cmap='bone')
ax2.set_title('Thresholded Fixed image')
ax3.imshow(dist_start_img, cmap='jet')
ax3.set_title('dmap Fixed Image')
ax4.imshow(dist_shift_img, cmap='jet')
ax4.set_title('dmap Moving Image');
```




```
# Start training
from matplotlib.animation import FuncAnimation
from IPython.display import HTML
import tensorflow as tf
from matplotlib import patches
optimize_iters = 20
loss_history = []
with tf.Session(graph=g_roi) as sess:
    plt.close('all')
    fig, m_axs = plt.subplots(2, 3, figsize=(12, 4), dpi=100)
    # Run the initializer
#    tf.initialize_all_variables().run()
    tf.global_variables_initializer()
    # Fit all training data
    const_feed_dict      = make_feed_dict(dist_start_img, dist_shift_img)
    real_image_feed_dict = make_feed_dict(bw_img, shift_img)

    def update_frame(i):
        global loss_history
        (ax1, ax2, ax5), (ax3, ax4, ax6) = m_axs
        for c_ax in m_axs.flatten():
            c_ax.cla()
            c_ax.axis('off')
        f_mse, x_pos, y_pos, rs_img, diff_img = sess.run([mse, x_offset, y_offset,␣
↪trans_tensor, diff_tensor],
                                                         feed_dict=const_feed_dict)
        real_rs_img, real_diff_img = sess.run([trans_tensor, diff_tensor],
                                              feed_dict=real_image_feed_dict)
```

(continues on next page)

(continued from previous page)

```
        loss_history += [f_mse]

        ax1.imshow(bw_img, cmap='bone')
        ax1.set_title('$T_0$')
        ax2.imshow(shift_img, cmap='bone')
        ax2.set_title('$T_1$')
        ax3.imshow(real_rs_img[0, :, :, 0], cmap='bone')
        ax3.set_title('Output')
        ax4.imshow(dist_start_img*1.0 -
                   rs_img[0, :, :, 0], cmap='RdBu', vmin=-10, vmax=10)
        ax4.set_title('MSE: %2.2f' % (f_mse))
        rect = patches.Rectangle(
            (25, 25), 75, 50, linewidth=2, edgecolor='g', facecolor='none')
        # Add the patch to the Axes
        ax4.add_patch(rect)
        ax5.semilogy(loss_history)
        ax5.set_xlabel('Iteration')
        ax5.set_ylabel('MSE\n(Log-scale)')
        ax5.axis('on')

        ax6.imshow(diff_img[0, :, :, 0], cmap='RdBu', vmin=-10, vmax=10)
        ax6.set_title('ROI')
        for _ in range(200):
            sess.run(optimizer, feed_dict=const_feed_dict)
    # write animation frames
    anim_code = FuncAnimation(fig,
                              update_frame,
                              frames=optimize_iters,
                              interval=1000,
                              repeat_delay=2000)
    anim_code.save('movies/smoothregistration_bone.mp4', fps=5, extra_args=['-vcodec',
↪ 'libx264'])
    plt.close('all')
```

```
%%file backport_ssim.py
from tensorflow.python.ops import array_ops, control_flow_ops, check_ops, math_ops,␣
↪nn_ops, nn
from tensorflow.python.framework import constant_op, dtypes, ops

# backporting new tensorflow ops is soo much fun
_SSIM_K1 = 0.01
_SSIM_K2 = 0.03


def _ssim_helper(x, y, reducer, max_val, compensation=1.0):
    r"""Helper function for computing SSIM.
    SSIM estimates covariances with weighted sums.  The default parameters
    use a biased estimate of the covariance:
    Suppose `reducer` is a weighted sum, then the mean estimators are
      \mu_x = \sum_i w_i x_i,
      \mu_y = \sum_i w_i y_i,
    where w_i's are the weighted-sum weights, and covariance estimator is
      cov_{xy} = \sum_i w_i (x_i - \mu_x) (y_i - \mu_y)
    with assumption \sum_i w_i = 1. This covariance estimator is biased, since
      E[cov_{xy}] = (1 - \sum_i w_i ^ 2) Cov(X, Y).
```

(continues on next page)

(continued from previous page)

```
    For SSIM measure with unbiased covariance estimators, pass as `compensation`
    argument (1 - \sum_i w_i ^ 2).
    Arguments:
      x: First set of images.
      y: Second set of images.
      reducer: Function that computes 'local' averages from set of images.
        For non-covolutional version, this is usually tf.reduce_mean(x, [1, 2]),
        and for convolutional version, this is usually tf.nn.avg_pool or
        tf.nn.conv2d with weighted-sum kernel.
      max_val: The dynamic range (i.e., the difference between the maximum
        possible allowed value and the minimum allowed value).
      compensation: Compensation factor. See above.
    Returns:
      A pair containing the luminance measure, and the contrast-structure measure.
    """
    c1 = (_SSIM_K1 * max_val) ** 2
    c2 = (_SSIM_K2 * max_val) ** 2

    # SSIM luminance measure is
    # (2 * mu_x * mu_y + c1) / (mu_x ** 2 + mu_y ** 2 + c1).
    mean0 = reducer(x)
    mean1 = reducer(y)
    num0 = mean0 * mean1 * 2.0
    den0 = math_ops.square(mean0) + math_ops.square(mean1)
    luminance = (num0 + c1) / (den0 + c1)

    # SSIM contrast-structure measure is
    #   (2 * cov_{xy} + c2) / (cov_{xx} + cov_{yy} + c2).
    # Note that `reducer` is a weighted sum with weight w_k, \sum_i w_i = 1, then
    #   cov_{xy} = \sum_i w_i (x_i - \mu_x) (y_i - \mu_y)
    #            = \sum_i w_i x_i y_i - (\sum_i w_i x_i) (\sum_j w_j y_j).
    num1 = reducer(x * y) * 2.0
    den1 = reducer(math_ops.square(x) + math_ops.square(y))
    c2 *= compensation
    cs = (num1 - num0 + c2) / (den1 - den0 + c2)

    # SSIM score is the product of the luminance and contrast-structure measures.
    return luminance, cs


def _fspecial_gauss(size, sigma):
    """Function to mimic the 'fspecial' gaussian MATLAB function."""
    size = ops.convert_to_tensor(size, dtypes.int32)
    sigma = ops.convert_to_tensor(sigma)

    coords = math_ops.cast(math_ops.range(size), sigma.dtype)
    coords -= math_ops.cast(size - 1, sigma.dtype) / 2.0

    g = math_ops.square(coords)
    g *= -0.5 / math_ops.square(sigma)

    g = array_ops.reshape(g, shape=[1, -1]) + \
        array_ops.reshape(g, shape=[-1, 1])
    g = array_ops.reshape(g, shape=[1, -1])  # For tf.nn.softmax().
    g = nn_ops.softmax(g)
    return array_ops.reshape(g, shape=[size, size, 1, 1])
```

(continues on next page)

(continued from previous page)

```
def _ssim_per_channel(img1, img2, max_val=1.0):
    """Computes SSIM index between img1 and img2 per color channel.
    This function matches the standard SSIM implementation from:
    Wang, Z., Bovik, A. C., Sheikh, H. R., & Simoncelli, E. P. (2004). Image
    quality assessment: from error visibility to structural similarity. IEEE
    transactions on image processing.
    Details:
      - 11x11 Gaussian filter of width 1.5 is used.
      - k1 = 0.01, k2 = 0.03 as in the original paper.
    Args:
      img1: First image batch.
      img2: Second image batch.
      max_val: The dynamic range of the images (i.e., the difference between the
        maximum the and minimum allowed values).
    Returns:
      A pair of tensors containing and channel-wise SSIM and contrast-structure
      values. The shape is [..., channels].
    """
    filter_size = constant_op.constant(11, dtype=dtypes.int32)
    filter_sigma = constant_op.constant(1.5, dtype=img1.dtype)

    shape1, shape2 = array_ops.shape_n([img1, img2])
    checks = [
        control_flow_ops.Assert(math_ops.reduce_all(math_ops.greater_equal(
            shape1[-3:-1], filter_size)), [shape1, filter_size], summarize=8),
        control_flow_ops.Assert(math_ops.reduce_all(math_ops.greater_equal(
            shape2[-3:-1], filter_size)), [shape2, filter_size], summarize=8)]

    # Enforce the check to run before computation.
    with ops.control_dependencies(checks):
        img1 = array_ops.identity(img1)

    # TODO(sjhwang): Try to cache kernels and compensation factor.
    kernel = _fspecial_gauss(filter_size, filter_sigma)
    kernel = array_ops.tile(kernel, multiples=[1, 1, shape1[-1], 1])

    # The correct compensation factor is `1.0 - tf.reduce_sum(tf.square(kernel))`,
    # but to match MATLAB implementation of MS-SSIM, we use 1.0 instead.
    compensation = 1.0

    # TODO(sjhwang): Try FFT.
    # TODO(sjhwang): Gaussian kernel is separable in space. Consider applying
    #   1-by-n and n-by-1 Gaussain filters instead of an n-by-n filter.
    def reducer(x):
        shape = array_ops.shape(x)
        x = array_ops.reshape(x, shape=array_ops.concat([[-1], shape[-3:]], 0))
        y = nn.depthwise_conv2d(
            x, kernel, strides=[1, 1, 1, 1], padding='VALID')
        return array_ops.reshape(y, array_ops.concat([shape[:-3],
                                                      array_ops.shape(y)[1:]], 0))

    luminance, cs = _ssim_helper(img1, img2, reducer, max_val, compensation)

    # Average over the second and the third from the last: height, width.
```

(continues on next page)

(continued from previous page)

```
    axes = constant_op.constant([-3, -2], dtype=dtypes.int32)
    ssim_val = math_ops.reduce_mean(luminance * cs, axes)
    cs = math_ops.reduce_mean(cs, axes)
    return ssim_val, cs
```

```
Overwriting backport_ssim.py
```

Try structural similarity index metric

We have used MSE as loss function, how about trying SSIM from lecture 3?

```
from backport_ssim import _ssim_per_channel
g_roi_ssim = tf.Graph()
with g_roi_ssim.as_default():
    init = tf.global_variables_initializer()
    # tf Graph Input
    fixed_img = tf.placeholder("float", shape=(
        1, None, None, 1), name='FixedImage')
    moving_img = tf.placeholder("float", shape=(
        1, None, None, 1), name='MovingImage')
    # Initialize the variables (i.e. assign their default value)

    with tf.name_scope('transform_parameters'):  # Set transform parameters
        x_offset = tf.Variable(0.0, name="x_offset")
        y_offset = tf.Variable(0.0, name="y_offset")
        # we keep rotation fixed
        rotation = tf.placeholder("float", shape=tuple(), name="rotation")

    with tf.name_scope('transformer_and_interpolator'):
        flat_mat = tf.tile([tf.cos(rotation), -tf.sin(rotation), x_offset,
                            tf.sin(rotation), tf.cos(rotation), y_offset], (1,))
        flat_mat = tf.reshape(flat_mat, (1, 6))
        trans_tensor = affine_transform(moving_img, flat_mat)

    with tf.name_scope('metric'):
        ssim, _ = _ssim_per_channel(fixed_img[:, 20:75, 25:100, :]/255.0,
                                    trans_tensor[:, 20:75, 25:100, :]/255.0,
                                    max_val=1.0)
        mssim = tf.reduce_mean(ssim, name='MeanSSIM')
        rev_mssim = 1-mssim  # since we can only minimize
        optimizer = tf.train.GradientDescentOptimizer(5e-2).minimize(rev_mssim)
```

```
# Start training
from matplotlib.animation import FuncAnimation
from IPython.display import HTML
from matplotlib import patches
optimize_iters = 40
loss_history = []
with tf.Session(graph=g_roi_ssim) as sess:
    plt.close('all')
    fig, m_axs = plt.subplots(2, 3, figsize=(15, 8), dpi=200)
    tf.initialize_all_variables().run()
    # Run the initializer
```

(continues on next page)

(continued from previous page)

```
    sess.run(init)
    # Fit all training data
    const_feed_dict = make_feed_dict(bw_img, shift_img)

    def update_frame(i):
        global loss_history
        (ax1, ax2, ax5), (ax3, ax4, ax6) = m_axs
        for c_ax in m_axs.flatten():
            c_ax.cla()
            c_ax.axis('off')
        f_ssim, x_pos, y_pos, rs_img = sess.run([mssim, x_offset, y_offset, trans_
↪tensor],
                                                feed_dict=const_feed_dict)
        loss_history += [f_ssim]

        ax1.imshow(bw_img, cmap='bone')
        ax1.set_title('$T_0$')
        ax2.imshow(shift_img, cmap='bone')
        ax2.set_title('$T_1$')
        ax3.imshow(rs_img[0, :, :, 0], cmap='bone')
        ax3.set_title('Output')
        ax4.imshow(bw_img*1.0-rs_img[0, :, :, 0],
                   cmap='RdBu', vmin=-100, vmax=100)
        ax4.set_title('Difference\nSSIM: %2.2f' % (f_ssim))
        rect = patches.Rectangle(
            (25, 20), 75, 55, linewidth=2, edgecolor='g', facecolor='none')
        # Add the patch to the Axes
        ax4.add_patch(rect)
        ax5.plot(loss_history)
        ax5.set_xlabel('Iteration')
        ax5.set_ylabel('SSIM')
        ax5.axis('on')

        for _ in range(1):
            sess.run(optimizer, feed_dict=const_feed_dict)
    # write animation frames
    anim_code = FuncAnimation(fig,
                              update_frame,
                              frames=optimize_iters,
                              interval=1000,
                              repeat_delay=2000)

    anim_code.save('movies/ssimregistration_bone.mp4', fps=5, extra_args=['-vcodec',
↪'libx264'])
    plt.close('all')
```

0.17.10 Registration using ITK and Simple ITK

For medical imaging the standard tools used are ITK and SimpleITK and they have been optimized over decades to deliver high-performance registration tasks. They are a bit clumsy to use from python, but they offer by far the best established tools for these problems.

[https://itk.org/ITKSoftwareGuide/html/Book2/ITKSoftwareGuide-Book2ch3.html]

Registration script

```
import SimpleITK as sitk

def register_img(fixed_arr,
                 moving_arr,
                 use_affine=True,
                 use_mse=True,
                 brute_force=True):
    fixed_image = sitk.GetImageFromArray(fixed_arr)
    moving_image = sitk.GetImageFromArray(moving_arr)
    transform = sitk.AffineTransform(
        2) if use_affine else sitk.ScaleTransform(2)
    initial_transform = sitk.CenteredTransformInitializer(sitk.Cast(fixed_image,␣
↪moving_image.GetPixelID()),
                                                          moving_image,
                                                          transform,
                                                          sitk.
↪CenteredTransformInitializerFilter.GEOMETRY)
    ff_img = sitk.Cast(fixed_image, sitk.sitkFloat32)
    mv_img = sitk.Cast(moving_image, sitk.sitkFloat32)
    registration_method = sitk.ImageRegistrationMethod()
    if use_mse:
        registration_method.SetMetricAsMeanSquares()
    else:
        registration_method.SetMetricAsMattesMutualInformation(
            numberOfHistogramBins=50)

    if brute_force:
        sample_per_axis = 12
        registration_method.SetOptimizerAsExhaustive(
            [sample_per_axis//2, 0, 0])
        # Utilize the scale to set the step size for each dimension
        registration_method.SetOptimizerScales(
            [2.0*3.14/sample_per_axis, 1.0, 1.0])
    else:
        registration_method.SetMetricSamplingStrategy(
            registration_method.RANDOM)
        registration_method.SetMetricSamplingPercentage(0.25)

    registration_method.SetInterpolator(sitk.sitkLinear)

    registration_method.SetOptimizerAsGradientDescent(learningRate=1.0,
                                                      numberOfIterations=200,
                                                      convergenceMinimumValue=1e-6,
                                                      convergenceWindowSize=10)
    # Scale the step size differently for each parameter, this is critical!!!
    registration_method.SetOptimizerScalesFromPhysicalShift()
```

(continues on next page)

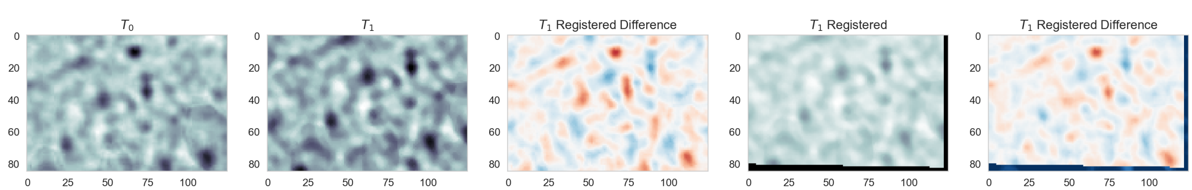
(continued from previous page)

```
    registration_method.SetInitialTransform(initial_transform, inPlace=False)
    final_transform_v1 = registration_method.Execute(ff_img,
                                                     mv_img)
    print('Optimizer\'s stopping condition, {0}'.format(
        registration_method.GetOptimizerStopConditionDescription()))
    print('Final metric value: {0}'.format(
        registration_method.GetMetricValue()))
    resample = sitk.ResampleImageFilter()
    resample.SetReferenceImage(fixed_image)

    # SimpleITK supports several interpolation options, we go with the simplest that gives reasonable results.
    resample.SetInterpolator(sitk.sitkBSpline)
    resample.SetTransform(final_transform_v1)
    return sitk.GetArrayFromImage(resample.Execute(moving_image))
```

Registration result

```
%matplotlib inline
reg_img = register_img(bw_img, shift_img, brute_force=False, use_mse=True)

print(reg_img.max(), bw_img.max())

fig, (ax1, ax2, ax2d, ax3, ax4) = plt.subplots(1, 5, figsize=(20, 5), dpi=100)
ax1.imshow(bw_img, cmap='bone')
ax1.set_title('$T_0$')
ax2.imshow(shift_img, cmap='bone')
ax2.set_title('$T_1$')
ax2d.imshow(1.0*bw_img-shift_img, cmap='RdBu', vmin=-100, vmax=100)
ax2d.set_title('$T_1$ Registered Difference')
ax3.imshow(reg_img, cmap='bone')
ax3.set_title('$T_1$ Registered')
ax4.imshow(1.0*bw_img-reg_img, cmap='RdBu', vmin=-127, vmax=127)
ax4.set_title('$T_1$ Registered Difference');#
```

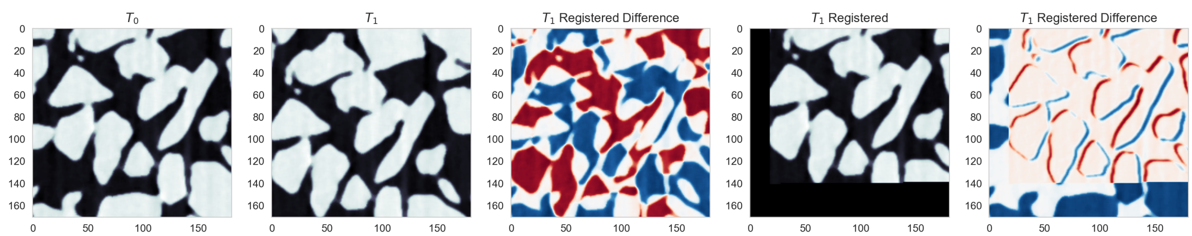
```
Optimizer's stopping condition, GradientDescentOptimizerv4Template: Convergence checker passed at iteration 28.
Final metric value: 206.73509032568484
161.43264033952298 165.0
```




```
%matplotlib inline
reg_img = register_img(bw_sand, shift_sand, brute_force=False, use_mse=True)

print(reg_img.max(), bw_img.max())
```

```
Optimizer's stopping condition, GradientDescentOptimizerv4Template: Convergence␣
↪checker passed at iteration 38.
Final metric value: 0.03965063534776068
0.9950255734897323 165.0
```

```
fig, (ax1, ax2, ax2d, ax3, ax4) = plt.subplots(1, 5, figsize=(20, 5), dpi=100)
ax1.imshow(bw_sand, cmap='bone')
ax1.set_title('$T_0$')
ax2.imshow(shift_sand, cmap='bone')
ax2.set_title('$T_1$')
ax2d.imshow(1.0*bw_sand-shift_sand, cmap='RdBu')
ax2d.set_title('$T_1$ Registered Difference')
ax3.imshow(reg_img, cmap='bone')
ax3.set_title('$T_1$ Registered')
ax4.imshow(1.0*bw_sand-reg_img, cmap='RdBu')
ax4.set_title('$T_1$ Registered Difference');#
```



0.17.11 Subdividing the data

We can approach the problem by subdividing the data into smaller blocks and then apply the digital volume correlation independently to each block.

- information on changes in different regions
- less statistics than a larger box

0.18 Introducing Physics

DIC or DVC by themselves include no *sanity check* for realistic offsets in the correlation itself. The method can, however be integrated with physical models to find a more optimal solutions.

- information from surrounding points
- smoothness criteria
- maximum deformation / force
- material properties

$$C_{\text{cost}} = \underbrace{C_{I_0,I_1}(\vec{r})}_{\text{Correlation Term}} + \underbrace{\lambda||\vec{r}||}_{\text{deformation term}}$$

0.18.1 Distribution Metrics

As we covered before distribution metrics like the distribution tensor can be used for tracking changes inside a sample. Of these the most relevant is the texture tensor from cellular materials and liquid foam. The texture tensor is the same as the distribution tensor except that the edges (or faces) represent physically connected / touching objects rather than touching Voronoi faces (or conversely Delaunay triangles).

These metrics can also be used for tracking the behavior of a system without tracking the single points since most deformations of a system also deform the distribution tensor and can thus be extracted by comparing the distribution tensor at different time steps.

0.18.2 Quantifying Deformation: Strain

We can take any of these approaches and quantify the deformation using a tool called the strain tensor.

Strain is defined in mechanics for the simple 1D case as the change in the length against the change in the original length.

$$e = \frac{\Delta L}{L}$$

While this defines the 1D case well, it is difficult to apply such metrics to voxel, shape, and tensor data.

0.18.3 Strain Tensor

There are a number of different ways to calculate strain and the strain tensor, but the most applicable for general image based applications is called the infinitesimal strain tensor, because the element matches well to square pixels and cubic voxels.

0.18.4 Types of Strain

We catagorize the types of strain into two main catagories:

$$\underbrace{\mathbf{E}}_{\text{Total Strain}} = \underbrace{\varepsilon_M \mathbf{I_3}}_{\text{Volumetric}} + \underbrace{\mathbf{E}'}_{\text{Deviatoric}}$$

Volumetric / Dilational

The isotropic change in size or scale of the object.

Deviatoric

The change in the proportions of the object (similar to anisotropy) independent of the final scale

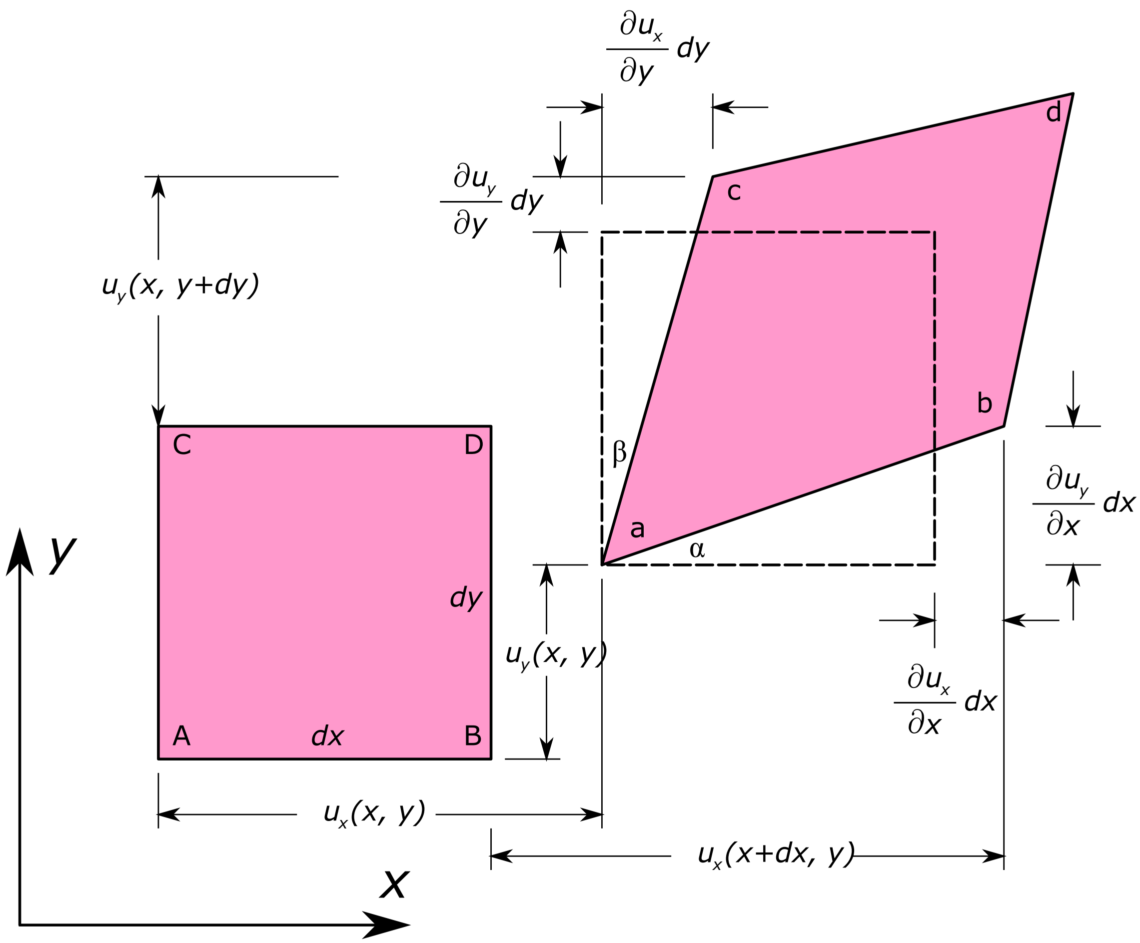


Fig. 1: Deformation by strain.

Shear band

0.19 Two Point Correlation - Volcanic Rock

Data provided by Mattia Pistone and Julie Fife

The air phase changes from small very anisotropic bubbles to one large connected pore network.

- The same tools cannot be used to quantify those systems.
- Furthermore there are motion artifacts which are difficult to correct.

We can utilize the two point correlation function of the material to characterize the shape generically for each time step and then compare.

0.20 Summary

- Dynamic experiments
- Object tracking
- Registration
- Digital volume correlation (DIC)