Advanced Astronomical Observations 2021
Session 19: Image Alignment

Kinoshita Daisuke

28 May 2021
publicly accessible version

About this file...

- Important information about this file
  - The author of this file is Kinoshita Daisuke.
  - The original version of this file was used for the course "Advanced Astronomical Observations" (course ID: AS6005) offered at Institute of Astronomy, National Central University from February 2021 to June 2021.
  - The file is provided in the hope that it will be useful, but there is no guarantee for the correctness. Use this file at your own risk.
  - If you are willing to use this file for your study, please feel free to use. I'll be very happy to receive feedback from you.
  - If you are willing to use this file for your teaching, please contact to Kinoshita Daisuke. When you use this file partly or entirely, please mention clearly that the author of the original version is Kinoshita Daisuke. Please help me to improve the contents of this file by sending your feedback.
  - Contact address: `https://www.instagram.com/daisuke23888/`

For this session, we try image alignment.

1 Python scripts for this session

All the Python scripts needed for this session are available at github.com. Visit the web page `https://github.com/kinoshitadaisuke/ncu_advobs_202102` to view files. (Fig. 1)

You may try following command to download all the Python scripts for this session.

```
% git clone https://github.com/kinoshitadaisuke/ncu_advobs_202102.git
Cloning into 'ncu_advobs_202102'...
remote: Enumerating objects: 183, done.
remote: Counting objects: 100% (183/183), done.
remote: Compressing objects: 100% (177/177), done.
remote: Total 183 (delta 96), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (183/183), 102.84 KiB | 877.00 KiB/s, done.
Resolving deltas: 100% (96/96), done.
% ls
ncu_advobs_202102/
% ls -l ncu_advobs_202102/
total 1
-rw-r--r--  1 daisuke  wheel  35149 May 28 01:09 LICENSE
-rw-r--r--  1 daisuke  wheel    568 May 28 01:09 README
-rw-r--r--  1 daisuke  wheel    343 May 28 01:09 REQUIREMENTS
-rw-r--r--  1 daisuke  wheel    342 May 28 01:09 download_python.csh
-rw-r--r--  1 daisuke  wheel    495 May 28 01:09 download_stds.csh
drwxr-xr-x  2 daisuke  wheel    512 May 28 01:09 s16_skybackground/
```

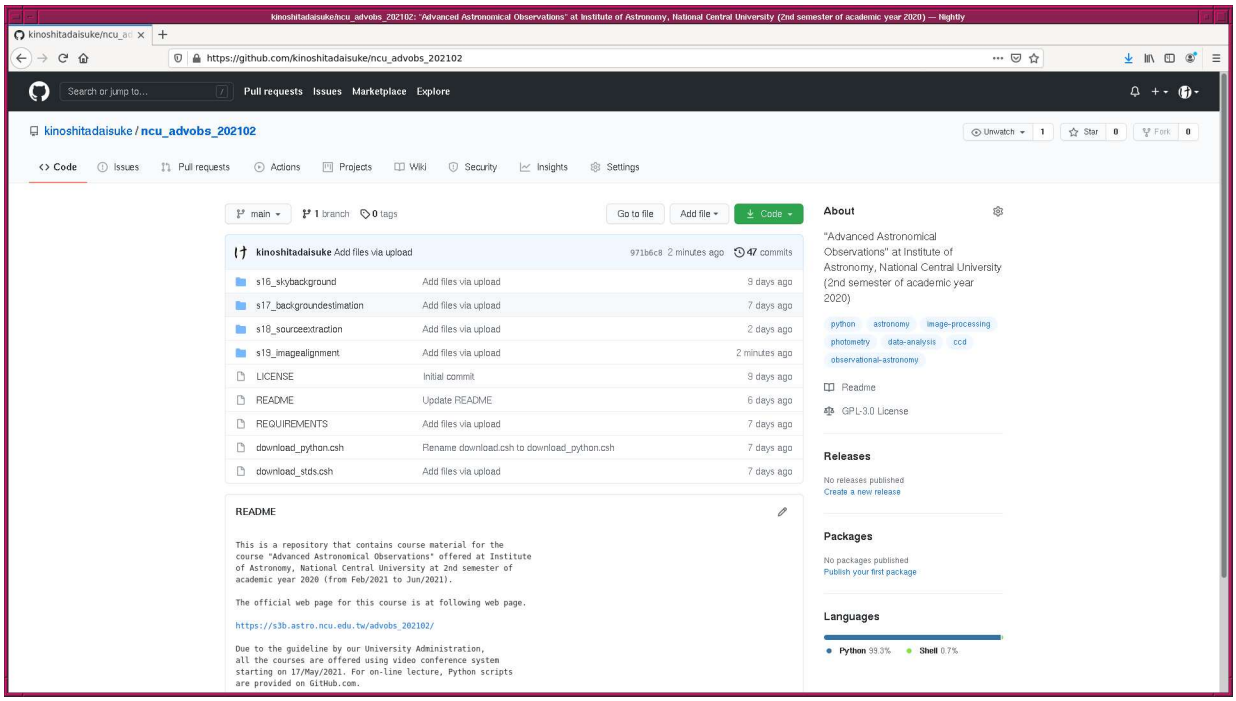

Figure 1: The GitHub repository for Python scripts for this course.

```
drwxr-xr-x  2 daisuke  wheel    512 May 28 01:09 s17_backgroundestimation/
drwxr-xr-x  2 daisuke  wheel    512 May 28 01:09 s18_sourceextraction/
drwxr-xr-x  2 daisuke  wheel    512 May 28 01:09 s19_imagealignment/
% ls -l ncu_advobs_202102/s19_imagealignment/
total 1
-rw-r--r--  1 daisuke  wheel     72 May 28 01:09 README
-rw-r--r--  1 daisuke  wheel   1977 May 28 01:09 ao2021_s19_01.py
-rw-r--r--  1 daisuke  wheel   2988 May 28 01:09 ao2021_s19_02.py
-rw-r--r--  1 daisuke  wheel   4451 May 28 01:09 ao2021_s19_03.py
-rw-r--r--  1 daisuke  wheel   6299 May 28 01:09 ao2021_s19_04.py
-rw-r--r--  1 daisuke  wheel   2956 May 28 01:09 ao2021_s19_05.py
-rw-r--r--  1 daisuke  wheel   6005 May 28 01:09 ao2021_s19_06.py
-rw-r--r--  1 daisuke  wheel   6592 May 28 01:09 ao2021_s19_07.py
-rw-r--r--  1 daisuke  wheel   3707 May 28 01:09 ao2021_s19_08.py
% head -15 ncu_advobs_202102/s19_imagealignment/ao2021_s19_01.py
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2021/05/27 15:22:52 (CST) daisuke>
#

# importing argparse module
import argparse

# importing sys module
import sys

# importing numpy module
import numpy.random
```

If above command fails on your computer, check whether or not you have the command `git`. You see something like following if you have the command `git` on your computer.

```
% which git
/usr/pkg/bin/git
```

You see following if you do not have the command `git` installed on your computer.

```
% which git
git: Command not found.
```

If you do not have the command `git`, install `git` on your computer.

2 Scikit-Image package

For this session, we need a package named `scikit-image`.

- scikit-image
  - `https://scikit-image.org/`

Try following to check whether or not you have `scikit-image` installed on your computer. If the package is properly installed on your computer, then you can successfully import `scikit-image`.

```
% python3.9
Python 3.9.2 (default, May  2 2021, 01:43:22)
[GCC 7.5.0] on netbsd9
Type "help", "copyright", "credits" or "license" for more information.
>>> import skimage
>>> exit ()
```

If you do not have `scikit-image` installed on your computer, then you see a message like below.

```
% python3.9
Python 3.9.2 (default, Mar 28 2021, 23:32:02)
[GCC 7.5.0] on netbsd9
Type "help", "copyright", "credits" or "license" for more information.
>>> import skimage
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'skimage'
>>> exit ()
```

3 Astroalign package

For this session, we need a package named `astroalign`.

- astroalign
  - `https://astroalign.readthedocs.io/`
  - `https://pypi.org/project/astroalign/`

Try following to check whether or not you have `astroalign` installed on your computer. If the package is properly installed on your computer, then you can successfully import `astroalign`.

```
% python3.9
Python 3.9.2 (default, May  2 2021, 01:43:22)
[GCC 7.5.0] on netbsd9
Type "help", "copyright", "credits" or "license" for more information.
>>> import astroalign
>>> exit ()
```

If you do not have `astroalign` installed on your computer, then you see a message like below.

```
% python3.9
Python 3.9.2 (default, Mar 27 2021, 17:26:25)
[GCC 7.5.0] on netbsd9
Type "help", "copyright", "credits" or "license" for more information.
>>> import astroalign
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'astroalign'
>>> exit ()
```

4 Generating a set of synthetic images

4.1 Generating (x, y) positions of stars

Make a Python script to generate (x, y) positions of artificial stars using `numpy.random` module.

Python Code 1: ao2021_s19_01.py

```python
#!/usr/pkg/bin/python3.9

# importing argparse module
import argparse

# importing sys module
import sys

# importing numpy module
import numpy.random

# constructing parser object
desc   = 'generating random (x, y) positions of stars'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-n', '--number', type=int, default=10, \
                     help='number of stars (default: 10)')
parser.add_argument ('-x', '--size-x', type=int, default=1024, \
                     help='image size on x-axis (default: 2048)')
parser.add_argument ('-y', '--size-y', type=int, default=1024, \
                     help='image size on y-axis (default: 2048)')
parser.add_argument ('-a', '--flux-min', type=float, default=100.0, \
                     help='minimum flux of stars (default: 100)')
parser.add_argument ('-b', '--flux-max', type=float, default=100000.0, \
                     help='maximum flux of stars (default: 100000)')
parser.add_argument ('-o', '--file-output', default='', \
                     help='output file name')
```

```
# command-line argument analysis
args = parser.parse_args ()

# parameters
nstars      = args.number
size_x      = args.size_x
size_y      = args.size_y
flux_min    = args.flux_min
flux_max    = args.flux_max
file_output = args.file_output

# check of output file name
if (file_output == ''):
    print ("Output file name must be specified.")
    sys.exit ()

# generating random numbers
position_x = numpy.random.default_rng ().uniform (0, size_x, nstars)
position_y = numpy.random.default_rng ().uniform (0, size_y, nstars)
flux       = numpy.random.default_rng ().uniform (flux_min, flux_max, nstars)

# writing data to file
with open (file_output, 'w') as fh_out:
    # for each object
    for i in range ( len (position_x) ):
        # writing x, y, flux to file
        fh_out.write ("%f %f %f\n" % (position_x[i], position_y[i], flux[i]) )
```

Run the script, and make a file.

```
% chmod a+x ao2021_s19_01.py
% ./ao2021_s19_01.py -h
usage: ao2021_s19_01.py [-h] [-n NUMBER] [-x SIZE_X] [-y SIZE_Y] [-a FLUX_MIN]
                        [-b FLUX_MAX] [-o FILE_OUTPUT]

generating random (x, y) positions of stars

optional arguments:
  -h, --help            show this help message and exit
  -n NUMBER, --number NUMBER
                        number of stars (default: 10)
  -x SIZE_X, --size-x SIZE_X
                        image size on x-axis (default: 2048)
  -y SIZE_Y, --size-y SIZE_Y
                        image size on y-axis (default: 2048)
  -a FLUX_MIN, --flux-min FLUX_MIN
                        minimum flux of stars (default: 100)
  -b FLUX_MAX, --flux-max FLUX_MAX
                        maximum flux of stars (default: 100000)
  -o FILE_OUTPUT, --file-output FILE_OUTPUT
                        output file name

% ./ao2021_s19_01.py -n 30 -x 2048 -y 2048 -o stars_0.list
% ls -l stars_0.list
-rw-r--r--  1 daisuke  taiwan  1073 May 27 15:29 stars_0.list
% head stars_0.list
1754.490563 533.916821 13276.806831
1891.687876 710.510552 95240.311316
```

```
1789.509525 1330.936712 61947.161879
930.549958 124.999272 38494.907232
36.833816 1678.899548 52529.746801
26.968957 928.833848 33644.089550
1797.306650 180.949460 50745.635328
1008.847047 103.187337 85585.390615
210.053864 958.555874 89562.132551
1220.961035 1724.396856 56153.990105
```

4.2 Rotation and translation of coordinates

Make a Python script to give rotation and translation of (x, y) coordinates.

Python Code 2: ao2021_s19_02.py

```
#!/usr/pkg/bin/python3.9

# importing argparse module
import argparse

# importing sys module
import sys

# importing numpy module
import numpy

# constructing parser object
desc   = 'rotation and translation of (x, y) coordinates'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-i', '--file-input', default='', \
                     help='input file name')
parser.add_argument ('-o', '--file-output', default='', \
                     help='output file name')
parser.add_argument ('-c', '--centre-x', type=float, default=1024.0, \
                     help='x-coordinate of centre of rotation (default: 1024)')
parser.add_argument ('-d', '--centre-y', type=float, default=1024.0, \
                     help='y-coordinate of centre of rotation (default: 1024)')
parser.add_argument ('-r', '--rotate', type=float, default=45.0, \
                     help='rotation angle in degree (default: 45)')
parser.add_argument ('-s', '--shift-x', type=float, default=10.0, \
                     help='amount of shift on x-axis (default:10)')
parser.add_argument ('-t', '--shift-y', type=float, default=10.0, \
                     help='amount of shift on y-axis (default:10)')

# command-line argument analysis
args = parser.parse_args ()

# parameters
file_input  = args.file_input
file_output = args.file_output
centre_x    = args.centre_x
centre_y    = args.centre_y
rotate_deg  = args.rotate
shift_x     = args.shift_x
shift_y     = args.shift_y

# conversion from deg to rad
```

```
rotate_rad = numpy.deg2rad (rotate_deg)

# check of input file name
if (file_input == ''):
    print ("Input file name must be specified.")
    sys.exit ()

# check of output file name
if (file_output == ''):
    print ("Output file name must be specified.")
    sys.exit ()

# list for new coordinates
list_new = []

# opening file for reading
with open (file_input, 'r') as fh_in:
    # reading file line-by-line
    for line in fh_in:
        # skipping line, if line starts with '#'
        if (line[0] == '#'):
            continue
        # splitting line
        (x_str, y_str, flux_str) = line.split ()
        # x, y, and flux
        x    = float (x_str)
        y    = float (y_str)
        flux = float (flux_str)
        # coordinate conversion (rotation and translation)
        x_0   = x - centre_x
        y_0   = y - centre_y
        x_1   = numpy.cos (rotate_rad) * x_0 - numpy.sin (rotate_rad) * y_0
        y_1   = numpy.sin (rotate_rad) * x_0 + numpy.cos (rotate_rad) * y_0
        x_2   = x_1 + shift_x
        y_2   = y_1 + shift_y
        x_new = x_2 + centre_x
        y_new = y_2 + centre_y
        # appending new positions to the list
        list_new.append ( (x_new, y_new, flux) )

# writing data to file
with open (file_output, 'w') as fh_out:
    # for each object
    for (x, y, flux) in list_new:
        # writing data
        fh_out.write ("%f %f %f\n" % (x, y, flux) )
```

Execute the script.

```
% chmod a+x ao2021_s19_02.py
% ./ao2021_s19_02.py -h
usage: ao2021_s19_02.py [-h] [-i FILE_INPUT] [-o FILE_OUTPUT] [-c CENTRE_X]
                        [-d CENTRE_Y] [-r ROTATE] [-s SHIFT_X] [-t SHIFT_Y]

rotation and translation of (x, y) coordinates

optional arguments:
  -h, --help            show this help message and exit
```

```
  -i FILE_INPUT, --file-input FILE_INPUT
                        input file name
  -o FILE_OUTPUT, --file-output FILE_OUTPUT
                        output file name
  -c CENTRE_X, --centre-x CENTRE_X
                        x-coordinate of centre of rotation (default: 1024)
  -d CENTRE_Y, --centre-y CENTRE_Y
                        y-coordinate of centre of rotation (default: 1024)
  -r ROTATE, --rotate ROTATE
                        rotation angle in degree (default: 45)
  -s SHIFT_X, --shift-x SHIFT_X
                        amount of shift on x-axis (default:10)
  -t SHIFT_Y, --shift-y SHIFT_Y
                        amount of shift on y-axis (default:10)

% ./ao2021_s19_02.py -i stars_0.list -o stars_1.list -r 15 -s 55 -t 30
% ls -l stars_*
-rw-r--r--  1 daisuke  taiwan  1073 May 27 15:29 stars_0.list
-rw-r--r--  1 daisuke  taiwan  1080 May 27 16:15 stars_1.list
% head stars_1.list
1911.442561 769.680870 13276.806831
1998.259168 975.766593 95240.311316
1738.984354 1548.606541 61947.161879
1221.412701 161.445328 38494.907232
-44.029788 1431.086978 52529.746801
140.572779 704.025934 33644.089550
2044.154401 439.822199 50745.635328
1302.687226 160.641395 85585.390615
309.726592 780.121067 89562.132551
1087.973705 1781.508679 56153.990105
```

4.3 Producing synthetic images

Make a Python script to read (x, y) coordinate file and produce a synthetic image.

Python Code 3: ao2021_s19_03.py

```
#!/usr/pkg/bin/python3.9

# importing argparse module
import argparse

# importing sys module
import sys

# importing numpy module
import numpy

# importing astropy module
import astropy.io
import astropy.table

# importing photutils module
import photutils.datasets

# constructing parser object
desc   = 'rotation and translation of (x, y) coordinates'
parser = argparse.ArgumentParser (description=desc)
```

```
# adding arguments
parser.add_argument ('-i', '--file-input', default='', \
                     help='input file name')
parser.add_argument ('-o', '--file-output', default='', \
                     help='output file name')
parser.add_argument ('-f', '--fwhm', type=float, default=4.0, \
                     help='FWHM of PSF in pixel (default: 4)')
parser.add_argument ('-g', '--fwhm-stddev', type=float, default=0.1, \
                     help='stddev of FWHM of PSF in pixel (default: 0.1)')
parser.add_argument ('-s', '--sky', type=float, default=1000.0, \
                     help='sky background level in ADU (default: 1000)')
parser.add_argument ('-e', '--sky-stddev', type=float, default=30.0, \
                     help='stddev of sky background in ADU (default: 30)')
parser.add_argument ('-x', '--size-x', type=int, default=2048, \
                     help='image size on x-axis (default: 2048)')
parser.add_argument ('-y', '--size-y', type=int, default=2048, \
                     help='image size on y-axis (default: 2048)')

# command-line argument analysis
args = parser.parse_args ()

# parameters
file_input  = args.file_input
file_output = args.file_output
fwhm        = args.fwhm
fwhm_stddev = args.fwhm_stddev
sky         = args.sky
sky_stddev  = args.sky_stddev
size_x      = args.size_x
size_y      = args.size_y

# image shape
image_shape = (size_x, size_y)

# check of input file name
if (file_input == ''):
    print ("Input file name must be specified.")
    sys.exit ()

# check of output file name
if not (file_output[-5:] == '.fits'):
    print ("Output file name must be a FITS file.")
    sys.exit ()

# generating a new astropy table
table_stars = astropy.table.Table ()

# making empty lists for data
list_x     = []
list_y     = []
list_flux  = []
list_psf_x = []
list_psf_y = []
list_theta = []

# opening file for reading
with open (file_input, 'r') as fh_in:
    # reading file line-by-line
    for line in fh_in:
```

```
        # skipping line, if line starts with '#'
        if (line[0] == '#'):
            continue
        # splitting line
        (x_str, y_str, flux_str) = line.split ()
        # x, y, and flux
        x    = float (x_str)
        y    = float (y_str)
        flux = float (flux_str)
        # random number generation for PSF
        psf_x = numpy.random.default_rng ().normal (loc=fwhm, \
                                                    scale=fwhm_stddev, \
                                                    size=1)
        psf_y = numpy.random.default_rng ().normal (loc=fwhm, \
                                                    scale=fwhm_stddev, \
                                                    size=1)
        theta_deg = numpy.random.default_rng ().uniform (0.0, 360.0, 1)
        theta_rad = numpy.deg2rad (theta_deg)
        # adding data to lists
        list_flux.append (flux)
        list_x.append (x)
        list_y.append (y)
        list_psf_x.append (psf_x)
        list_psf_y.append (psf_y)
        list_theta.append (theta_rad)

# adding data to astropy table
table_stars['amplitude'] = list_flux
table_stars['x_mean']    = list_x
table_stars['y_mean']    = list_y
table_stars['x_stddev']  = list_psf_x
table_stars['y_stddev']  = list_psf_y
table_stars['theta']     = list_theta

# generating stars
image_stars = photutils.datasets.make_gaussian_sources_image (image_shape, \
                                                              table_stars)
# generating sky background
image_sky = photutils.datasets.make_noise_image (image_shape, \
                                                 distribution='gaussian', \
                                                 mean=sky, \
                                                 stddev=sky_stddev)
# generating synthetic image
image = image_stars + image_sky

# making WCS
wcs = photutils.datasets.make_wcs (image_shape)
# making Image HDU
hdu = photutils.datasets.make_imagehdu (image, wcs=wcs)
# writing a FITS file
hdu.writeto (file_output)
```

Run the script, and make FITS files.

```
% chmod a+x ao2021_s19_03.py
% ./ao2021_s19_03.py -h
usage: ao2021_s19_03.py [-h] [-i FILE_INPUT] [-o FILE_OUTPUT] [-f FWHM]
                        [-g FWHM_STDDEV] [-s SKY] [-e SKY_STDDEV] [-x SIZE_X]
```

```
                        [-y SIZE_Y]

rotation and translation of (x, y) coordinates

optional arguments:
  -h, --help            show this help message and exit
  -i FILE_INPUT, --file-input FILE_INPUT
                        input file name
  -o FILE_OUTPUT, --file-output FILE_OUTPUT
                        output file name
  -f FWHM, --fwhm FWHM  FWHM of PSF in pixel (default: 4)
  -g FWHM_STDDEV, --fwhm-stddev FWHM_STDDEV
                        stddev of FWHM of PSF in pixel (default: 0.1)
  -s SKY, --sky SKY     sky background level in ADU (default: 1000)
  -e SKY_STDDEV, --sky-stddev SKY_STDDEV
                        stddev of sky background in ADU (default: 30)
  -x SIZE_X, --size-x SIZE_X
                        image size on x-axis (default: 2048)
  -y SIZE_Y, --size-y SIZE_Y
                        image size on y-axis (default: 2048)

% ./ao2021_s19_03.py -i stars_0.list -o stars_0.fits -f 3.5 -s 1600 -e 40
% ./ao2021_s19_03.py -i stars_1.list -o stars_1.fits -f 4.5 -s 2500 -e 50
% ls -l *.fits
-rw-r--r--  1 daisuke  taiwan  33560640 May 27 16:48 stars_0.fits
-rw-r--r--  1 daisuke  taiwan  33560640 May 27 16:48 stars_1.fits
```

4.4 Visual inspection of FITS files

Use Ginga to check the FITS files. (Fig. 2 and 3)

```
% ginga stars_?.fits
```

5 Source extraction

Make a Python script to carry out source extraction using image segmentation.

Python Code 4: ao2021_s19_04.py

```python
#!/usr/pkg/bin/python3.9

# importing argparse module
import argparse

# importing sys module
import sys

# importing datetime module
import datetime

# importing numpy module
import numpy.ma

# importing astropy module
import astropy.convolution
import astropy.io.fits
```

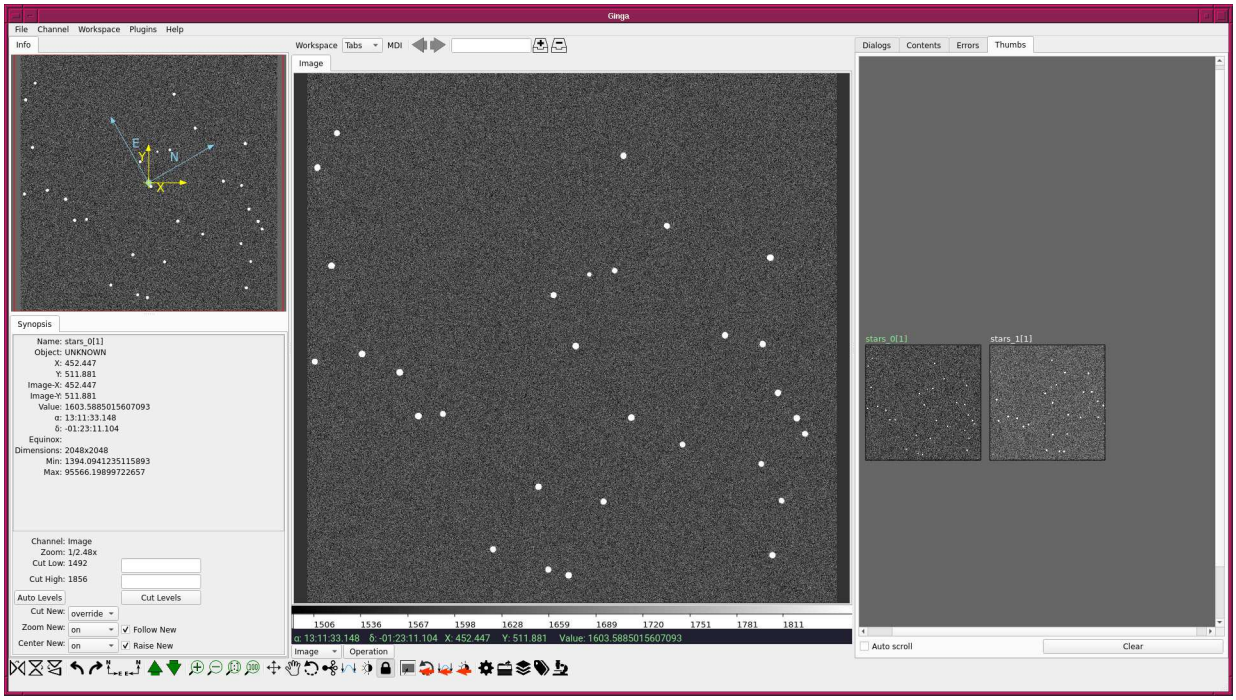

Figure 2: First synthetic image.

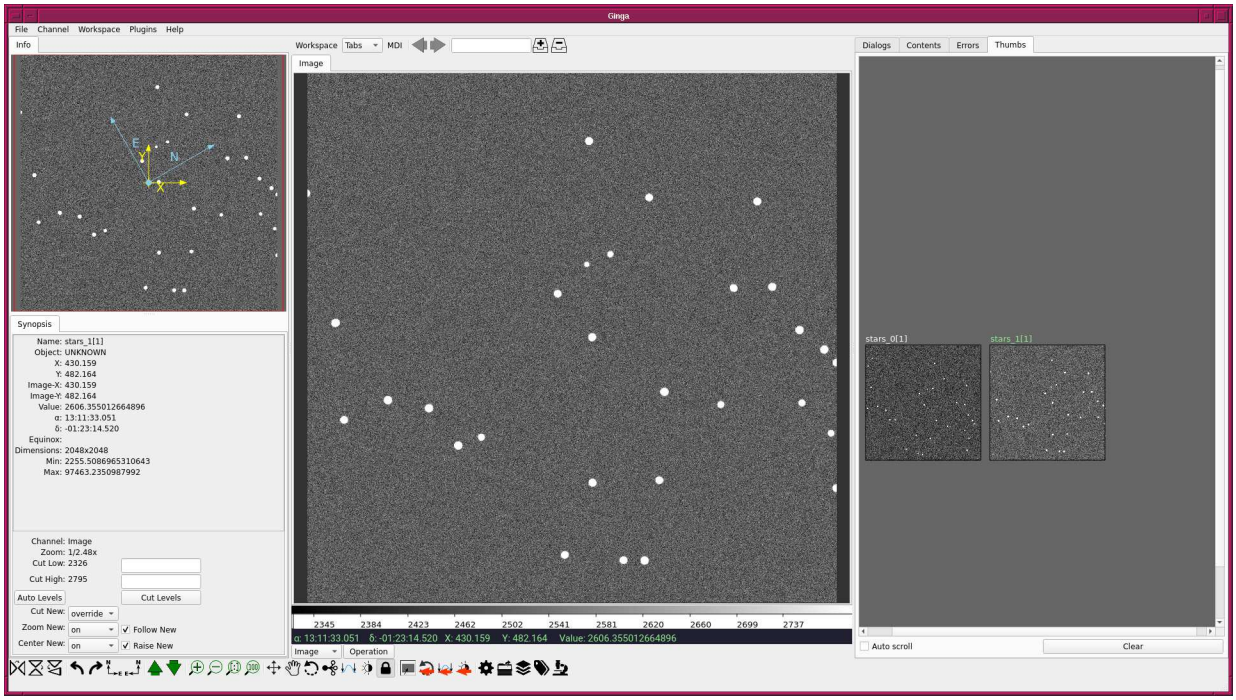

Figure 3: Second synthetic image.

```
import astropy.stats
import astropy.visualization
import astropy.visualization.mpl_normalize

# importing photutils module
import photutils.segmentation

# importing matplotlib module
import matplotlib.figure
import matplotlib.backends.backend_agg

# date/time
now = datetime.datetime.now ()

# constructing parser object
desc   = 'source extraction using image segmentation and deblending'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-i', '--file-input', default='', \
                     help='input FITS file name')
parser.add_argument ('-o', '--file-catalogue', default='', \
                     help='output catalogue file name')
parser.add_argument ('-g', '--file-fig', default='', \
                     help='output figure file name')
parser.add_argument ('-t', '--threshold', type=float, default=3.0, \
                     help='detection threshold in sigma (default: 3)')
parser.add_argument ('-u', '--threshold-for-sky', type=float, default=2.0, \
                     help='detection threshold for sky estimate (default: 2)')
parser.add_argument ('-n', '--npixels', type=int, default=5, \
                     help='minimum number of pixels for detection (default: 5)')
parser.add_argument ('-s', '--dilate-size', type=int, default=21, \
                     help='dilate size (default: 21)')
parser.add_argument ('-m', '--maxiters', type=int, default=30, \
                     help='maximum number of iterations (default: 30)')
parser.add_argument ('-c', '--sigma-clipping', type=float, default=4.0, \
                     help='sigma-clipping threshold in sigma (default: 4)')
parser.add_argument ('-k', '--gaussian-fwhm', type=float, default=3.0, \
                     help='Gaussian FWHM in pixel for convolution (default: 3)')
parser.add_argument ('-a', '--kernel-size', type=int, default=3, \
                     help='Gaussian kernel array size in pixel (default: 3)')
parser.add_argument ('-r', '--radius', type=float, default=30.0, \
                     help='radius of aperture in pixel (default: 30)')

# command-line argument analysis
args = parser.parse_args ()

# file names
file_input     = args.file_input
file_catalogue = args.file_catalogue
file_fig       = args.file_fig

# input parameters
threshold         = args.threshold
threshold_for_sky = args.threshold
npixels           = args.npixels
dilate_size       = args.dilate_size
maxiters          = args.maxiters
rejection         = args.sigma_clipping
```

```
gaussian_fwhm     = args.gaussian_fwhm
kernel_array_size = args.kernel_size
radius            = args.radius

# check of input file name
if not (file_input[-5:] == '.fits'):
    print ("Input file must be a FITS file.")
    sys.exit ()

# check of catalogue file name
if (file_catalogue == ''):
    print ("Catalogue file name must be specified.")
    sys.exit ()

# check of figure file name
if not ( (file_fig[-4:] == '.eps') or (file_fig[-4:] == '.pdf') \
         or (file_fig[-4:] == '.png') or (file_fig[-3:] == '.ps') ):
    print ("Figure file name must be either EPS, PDF, PNG, or PS.")
    sys.exit ()

# opening FITS file
with astropy.io.fits.open (file_input) as hdu:
    # reading header and image
    header = hdu[0].header
    image  = hdu[0].data
    # if no image in PrimaryHDU, then read next HDU
    if (header['NAXIS'] == 0):
        header = hdu[1].header
        image  = hdu[1].data

# making source mask
source_mask \
    = photutils.segmentation.make_source_mask (image, threshold_for_sky,
                                               npixels=npixels,
                                               sigclip_iters=maxiters,
                                               dilate_size=dilate_size)

# making masked array
image_masked = numpy.ma.array (image, mask=source_mask)

# sigma-clipping
skybg_mean, skybg_median, skybg_stddev \
    = astropy.stats.sigma_clipped_stats (image, sigma=rejection)

# mode calculation using empirical formula
skybg_mode = 3.0 * skybg_median - 2.0 * skybg_mean

# detection threshold in ADU
threshold_adu = skybg_mode + threshold * skybg_stddev

# 2D Gaussian kernel for convolution
gaussian_sigma = gaussian_fwhm * astropy.stats.gaussian_fwhm_to_sigma
kernel = astropy.convolution.Gaussian2DKernel (gaussian_sigma, \
                                               x_size=kernel_array_size, \
                                               y_size=kernel_array_size)
kernel.normalize ()

# source detection
image_segm = photutils.segmentation.detect_sources (image, threshold_adu, \
```

```
                                                    npixels=npixels, \
                                                    filter_kernel=kernel)

# deblending
image_deblend = photutils.segmentation.deblend_sources (image, image_segm, \
                                                        npixels=npixels, \
                                                        filter_kernel=kernel, \
                                                        nlevels=32, \
                                                        contrast=0.001)

# making a source catalogue
catalogue = photutils.segmentation.SourceCatalog (image, image_deblend)

# making a table
table_source = catalogue.to_table ()

# writing table to a file
astropy.io.ascii.write (table_source, file_catalogue, format='commented_header')

# positions of apertures
list_x = list (table_source['xcentroid'])
list_y = list (table_source['ycentroid'])
positions = []
for i in range ( len (list_x) ):
    positions.append ( (list_x[i], list_y[i]) )

# apertures
apertures = photutils.aperture.CircularAperture (positions, r=radius)

# matplotlib
norm \
    = astropy.visualization.mpl_normalize.ImageNormalize \
    ( stretch=astropy.visualization.HistEqStretch (image) )
matplotlib.pyplot.imshow (image, origin='lower', cmap='viridis', norm=norm)
apertures.plot (color='red', lw=1.0, alpha=0.5)
matplotlib.pyplot.title ("Detected sources")
matplotlib.pyplot.savefig (file_fig, dpi=225)
```

Execute the script, and generate a list of sources detected from the image.

```
% chmod a+x ao2021_s19_04.py
% ./ao2021_s19_04.py -h
usage: ao2021_s19_04.py [-h] [-i FILE_INPUT] [-o FILE_CATALOGUE] [-g FILE_FIG]
                        [-t THRESHOLD] [-u THRESHOLD_FOR_SKY] [-n NPIXELS]
                        [-s DILATE_SIZE] [-m MAXITERS] [-c SIGMA_CLIPPING]
                        [-k GAUSSIAN_FWHM] [-a KERNEL_SIZE] [-r RADIUS]

source extraction using image segmentation and deblending

optional arguments:
  -h, --help            show this help message and exit
  -i FILE_INPUT, --file-input FILE_INPUT
                        input FITS file name
  -o FILE_CATALOGUE, --file-catalogue FILE_CATALOGUE
                        output catalogue file name
  -g FILE_FIG, --file-fig FILE_FIG
                        output figure file name
  -t THRESHOLD, --threshold THRESHOLD
```

```
                        detection threshold in sigma (default: 3)
  -u THRESHOLD_FOR_SKY, --threshold-for-sky THRESHOLD_FOR_SKY
                        detection threshold for sky estimate (default: 2)
  -n NPIXELS, --npixels NPIXELS
                        minimum number of pixels for detection (default: 5)
  -s DILATE_SIZE, --dilate-size DILATE_SIZE
                        dilate size (default: 21)
  -m MAXITERS, --maxiters MAXITERS
                        maximum number of iterations (default: 30)
  -c SIGMA_CLIPPING, --sigma-clipping SIGMA_CLIPPING
                        sigma-clipping threshold in sigma (default: 4)
  -k GAUSSIAN_FWHM, --gaussian-fwhm GAUSSIAN_FWHM
                        Gaussian FWHM in pixel for convolution (default: 3)
  -a KERNEL_SIZE, --kernel-size KERNEL_SIZE
                        Gaussian kernel array size in pixel (default: 3)
  -r RADIUS, --radius RADIUS
                        radius of aperture in pixel (default: 30)

% ./ao2021_s19_04.py -i stars_0.fits -o stars_0.cat -g stars_0.pdf
% ./ao2021_s19_04.py -i stars_1.fits -o stars_1.cat -g stars_1.pdf
% ls -l stars_?.cat stars_?.pdf
-rw-r--r--  1 daisuke  taiwan     7068 May 27 17:40 stars_0.cat
-rw-r--r--  1 daisuke  taiwan  1677899 May 27 17:40 stars_0.pdf
-rw-r--r--  1 daisuke  taiwan     6859 May 27 17:41 stars_1.cat
-rw-r--r--  1 daisuke  taiwan  1675512 May 27 17:40 stars_1.pdf
% head -3 stars_?.cat
==> stars_0.cat <==
# label xcentroid ycentroid sky_centroid bbox_xmin bbox_xmax bbox_ymin bbox_ymax
 area semimajor_sigma semiminor_sigma orientation eccentricity min_value max_val
ue local_background segment_flux segment_fluxerr kron_flux kron_fluxerr
1 1008.8435314930749 103.19271049928068 None 996 1021 91 116 522.0 3.93058848438
3551 3.912368788380299 49.1477749675409 0.09617294392184099 1637.4747835161468 8
7039.70776482752 0.0 7337657.54870799 nan 8395446.04477655 nan
2 930.5437886214696 125.00770111816485 None 919 942 113 137 461.0 4.169466128296
234 4.065716001458044 -50.7501025360275 0.2216922301786795 1634.4811857422683 39
775.28682591826 0.0 3673677.494503547 nan 6438319.680325905 nan

==> stars_1.cat <==
# label xcentroid ycentroid sky_centroid bbox_xmin bbox_xmax bbox_ymin bbox_ymax
 area semimajor_sigma semiminor_sigma orientation eccentricity min_value max_val
ue local_background segment_flux segment_fluxerr kron_flux kron_fluxerr
1 1302.702121563132 160.62719475513978 None 1287 1318 145 176 811.0 5.2558650091
467705 5.096672616209971 -81.23537903147121 0.24425325610324652 2555.82417566517
9 87514.60765097433 0.0 12621151.64258882 nan 17526495.013319142 nan
2 1221.437263670592 161.46136148186312 None 1207 1237 147 176 713.0 5.5899124677
32806 5.305399991696487 -1.3685936485764667 0.31496731996319405 2545.46692742708
23 40685.78036749206 0.0 6613371.455324532 nan 17210123.43057765 nan
```

Display PDF files, and check the results of source extraction. (Fig. 4 and 5)

```
% xpdf stars_0.pdf
% xpdf stars_1.pdf
```

6 Finding star-to-star correspondence

Make a Python script to find star-to-star correspondence of two images.

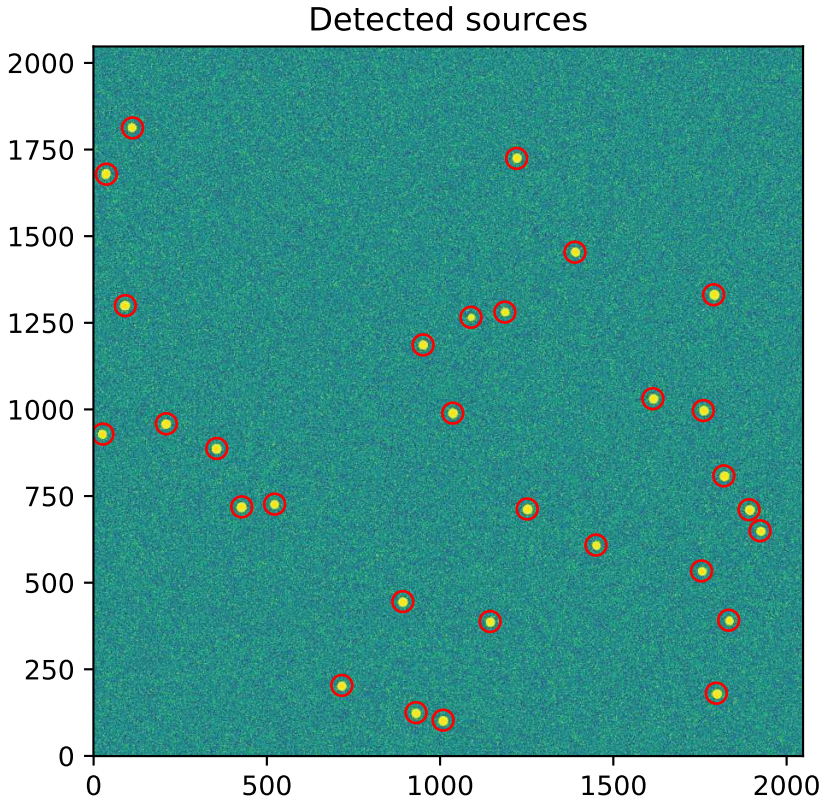

Figure 4: Sources detected from the FITS file `stars_0.fits`.

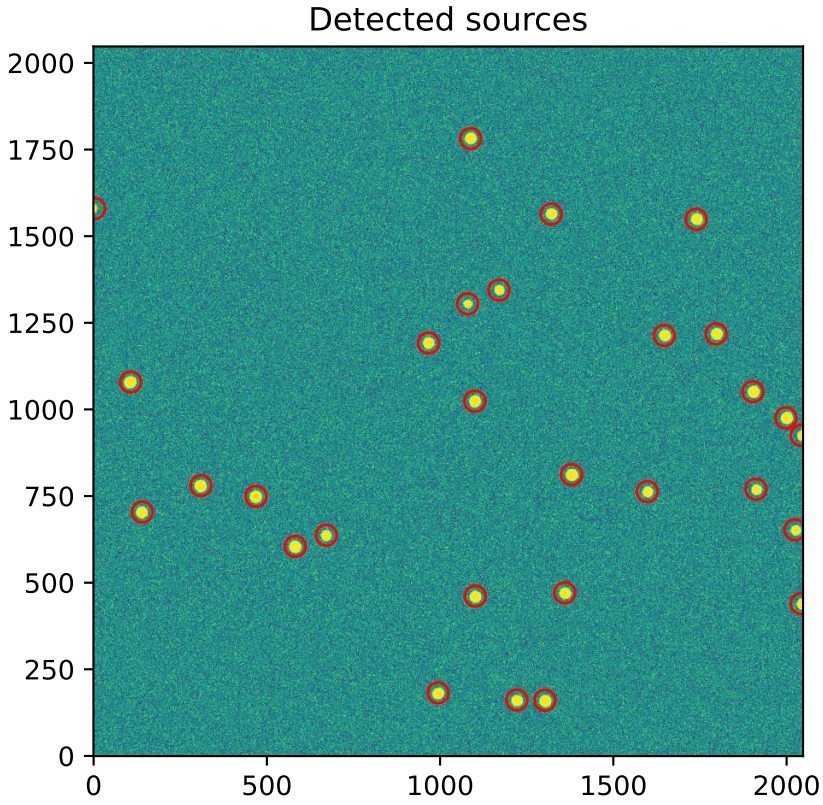

Figure 5: Sources detected from the FITS file `stars_1.fits`.

Python Code 5: ao2021_s19_05.py

```
#!/usr/pkg/bin/python3.9

# importing argparse module
import argparse

# importing sys module
import sys

# importing datetime module
import datetime

# importing numpy module
import numpy

# importing astropy module
import astropy.table

# importing astroalign module
import astroalign

# date/time
now = datetime.datetime.now ()

# constructing parser object
desc   = 'finding star-to-star correspondence'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('catalogue1', nargs=1, help='catalogue file 1')
parser.add_argument ('catalogue2', nargs=1, help='catalogue file 2')

# command-line argument analysis
args = parser.parse_args ()

# file names
file_cat1 = args.catalogue1[0]
file_cat2 = args.catalogue2[0]

# check of catalogue file name
if not ( (file_cat1[-4:] == '.cat') and (file_cat2[-4:] == '.cat') ):
    print ("Input file must be a catalogue file (*.cat).")
    print ("catalogue file 1 = %s" % file_cat1)
    print ("catalogue file 2 = %s" % file_cat2)
    sys.exit ()

# reading catalogue from a file
table_source1 = astropy.table.Table.read (file_cat1, \
                                          format='ascii.commented_header')
table_source2 = astropy.table.Table.read (file_cat2, \
                                          format='ascii.commented_header')

# (x, y) coordinates of sources
list_source1_x = list (table_source1['xcentroid'])
list_source1_y = list (table_source1['ycentroid'])
list_source2_x = list (table_source2['xcentroid'])
list_source2_y = list (table_source2['ycentroid'])
position_1 = numpy.transpose ( (list_source1_x, list_source1_y) )
position_2 = numpy.transpose ( (list_source2_x, list_source2_y) )
```

```
# finding star-to-star matching
transf, (list_matched_2, list_matched_1) \
    = astroalign.find_transform (position_2, position_1)

# transformation
list_matched_2_aligned \
    = astroalign.matrix_transform (list_matched_2, transf.params)

# printing results
print ("#")
print ("# result of image alignment")
print ("#")
print ("#   date/time = %s" % now)
print ("#")
print ("# input files")
print ("#")
print ("#   catalogue file 1 = %s" % file_cat1)
print ("#   catalogue file 2 = %s" % file_cat2)
print ("#")
print ("# transformation matrix")
print ("#")
print ("# [")
print ("#   [%f, %f, %f]," \
       % (transf.params[0][0], transf.params[0][1], transf.params[0][2]) )
print ("#   [%f, %f, %f]," \
       % (transf.params[1][0], transf.params[1][1], transf.params[1][2]) )
print ("#   [%f, %f, %f]" \
       % (transf.params[2][0], transf.params[2][1], transf.params[2][2]) )
print ("# ]")
print ("#")
print ("#")
print ("# list of matched stars")
print ("#")
for i in range ( len (list_matched_1) ):
    print ("(%10.4f, %10.4f) on 1st image ==> (%10.4f, %10.4f) on 2nd image" \
           % (list_matched_1[i][0], list_matched_1[i][1], \
              list_matched_2[i][0], list_matched_2[i][1]) )
```

Run the script, and carry out matching.

```
% chmod a+x ao2021_s19_05.py
% ./ao2021_s19_05.py -h
usage: ao2021_s19_05.py [-h] catalogue1 catalogue2

finding star-to-star correspondence

positional arguments:
  catalogue1  catalogue file 1
  catalogue2  catalogue file 2

optional arguments:
  -h, --help  show this help message and exit

% ./ao2021_s19_05.py stars_0.cat stars_1.cat
#
# result of image alignment
#
```

```
#   date/time = 2021-05-27 19:39:02.962335
#
# input files
#
#   catalogue file 1 = stars_0.cat
#   catalogue file 2 = stars_1.cat
#
# transformation matrix
#
# [
#   [0.965925, 0.258817, -291.029791],
#   [-0.258817, 0.965925, 285.177948],
#   [0.000000, 0.000000, 1.000000]
# ]
#
#
# list of matched stars
#
(  427.6536,   718.7830) on 1st image ==> (  581.9708,   604.8288) on 2nd image
( 1754.4875,   533.9240) on 1st image ==> ( 1911.4250,   769.6463) on 2nd image
( 1144.3900,   387.9610) on 1st image ==> ( 1359.9034,   470.8140) on 2nd image
(  892.4160,   445.5007) on 1st image ==> ( 1101.6212,   461.1451) on 2nd image
( 1089.1050,  1265.9480) on 1st image ==> ( 1079.3278,  1304.7173) on 2nd image
( 1036.5090,   989.3659) on 1st image ==> ( 1100.0430,  1023.8028) on 2nd image
( 1449.8699,   608.5264) on 1st image ==> ( 1597.8592,   762.8984) on 2nd image
(  355.5129,   887.3405) on 1st image ==> (  468.6425,   748.9694) on 2nd image
(   91.5945,  1299.5430) on 1st image ==> (  107.0605,  1078.8181) on 2nd image
( 1891.6832,   710.5214) on 1st image ==> ( 1998.2563,   975.7711) on 2nd image
(  210.0494,   958.5712) on 1st image ==> (  309.7215,   780.1374) on 2nd image
( 1187.0417,  1280.8644) on 1st image ==> ( 1170.1186,  1344.1679) on 2nd image
( 1789.5059,  1330.9291) on 1st image ==> ( 1738.9862,  1548.6088) on 2nd image
(  522.5328,   726.6528) on 1st image ==> (  671.5495,   636.9774) on 2nd image
( 1389.2390,  1453.7359) on 1st image ==> ( 1320.5294,  1563.6546) on 2nd image
( 1759.6721,   996.8308) on 1st image ==> ( 1796.6302,  1218.1563) on 2nd image
( 1832.8962,   391.6778) on 1st image ==> ( 2024.0220,   652.5956) on 2nd image
(   26.9681,   928.8527) on 1st image ==> (  140.5583,   704.0075) on 2nd image
(  950.9102,  1186.2260) on 1st image ==> (  966.4180,  1191.7817) on 2nd image
( 1251.5349,   712.6292) on 1st image ==> ( 1379.3614,   812.1382) on 2nd image
( 1614.0986,  1031.0432) on 1st image ==> ( 1647.1573,  1213.5161) on 2nd image
( 1008.8435,   103.1927) on 1st image ==> ( 1302.7021,   160.6272) on 2nd image
(  930.5438,   125.0077) on 1st image ==> ( 1221.4373,   161.4614) on 2nd image
(  716.4430,   203.5878) on 1st image ==> (  994.2842,   181.9682) on 2nd image
( 1220.9639,  1724.3945) on 1st image ==> ( 1087.9755,  1781.5115) on 2nd image
( 1818.8588,   808.2168) on 1st image ==> ( 1902.6105,  1051.2960) on 2nd image
```

7 Check of star-to-star correspondence

Make a Python script to check star-to-star correspondence.

Python Code 6: ao2021_s19_06.py

```
#!/usr/pkg/bin/python3.9

# importing argparse module
import argparse

# importing sys module
import sys
```

```
# importing datetime module
import datetime

# importing numpy module
import numpy

# importing astropy module
import astropy.table
import astropy.visualization

# importing astroalign module
import astroalign

# importing matplotlib module
import matplotlib.figure
import matplotlib.backends.backend_agg

# date/time
now = datetime.datetime.now ()

# constructing parser object
desc   = 'finding star-to-star correspondence'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-o', '--file-output', default='', \
                     help='output figure file')
parser.add_argument ('catalogue1', nargs=1, help='catalogue file 1')
parser.add_argument ('catalogue2', nargs=1, help='catalogue file 2')
parser.add_argument ('fits1', nargs=1, help='FITS file 1')
parser.add_argument ('fits2', nargs=1, help='FITS file 2')

# command-line argument analysis
args = parser.parse_args ()

# file names
file_cat1  = args.catalogue1[0]
file_cat2  = args.catalogue2[0]
file_fits1 = args.fits1[0]
file_fits2 = args.fits2[0]
file_fig   = args.file_output

# check of output file name
if not ( (file_fig[-4:] == '.eps') or (file_fig[-4:] == '.pdf') \
         or (file_fig[-4:] == '.png') or (file_fig[-3:] == '.ps') ):
    print ("Figure file name must be either EPS, PDF, PNG, or PS.")
    sys.exit ()

# check of catalogue file name
if not ( (file_cat1[-4:] == '.cat') and (file_cat2[-4:] == '.cat') ):
    print ("Input file must be a catalogue file (*.cat).")
    print ("catalogue file 1 = %s" % file_cat1)
    print ("catalogue file 2 = %s" % file_cat2)
    sys.exit ()

# check of FITS file name
if not ( (file_fits1[-5:] == '.fits') and (file_fits2[-5:] == '.fits') ):
    print ("Input file must be a FITS file (*.fits).")
```

```
    print ("FITS file 1 = %s" % file_fits1)
    print ("FITS file 2 = %s" % file_fits2)
    sys.exit ()

# function to read a FITS file
def read_fits (file_fits):
    # reading FITS file
    with astropy.io.fits.open (file_fits) as hdu:
        # reading header and image
        header = hdu[0].header
        image  = hdu[0].data
        # if no image in PrimaryHDU, then read next HDU
        if (header['NAXIS'] == 0):
            header = hdu[1].header
            image  = hdu[1].data
    # returning header and image
    return (header, image)

# reading catalogue from a file
table_source1 = astropy.table.Table.read (file_cat1, \
                                          format='ascii.commented_header')
table_source2 = astropy.table.Table.read (file_cat2, \
                                          format='ascii.commented_header')

# (x, y) coordinates of sources
list_source1_x = list (table_source1['xcentroid'])
list_source1_y = list (table_source1['ycentroid'])
list_source2_x = list (table_source2['xcentroid'])
list_source2_y = list (table_source2['ycentroid'])
position_1 = numpy.transpose ( (list_source1_x, list_source1_y) )
position_2 = numpy.transpose ( (list_source2_x, list_source2_y) )

# finding star-to-star matching
transf, (list_matched_2, list_matched_1) \
    = astroalign.find_transform (position_2, position_1)

# transformation
list_matched_2_aligned \
    = astroalign.matrix_transform (list_matched_2, transf.params)

# printing results
print ("#")
print ("# result of image alignment")
print ("#")
print ("#   date/time = %s" % now)
print ("#")
print ("# input files")
print ("#")
print ("#   catalogue file 1 = %s" % file_cat1)
print ("#   catalogue file 2 = %s" % file_cat2)
print ("#")
print ("# transformation matrix")
print ("#")
print ("# [")
print ("#   [%f, %f, %f]," \
       % (transf.params[0][0], transf.params[0][1], transf.params[0][2]) )
print ("#   [%f, %f, %f]," \
       % (transf.params[1][0], transf.params[1][1], transf.params[1][2]) )
print ("#   [%f, %f, %f]" \
```

```
       % (transf.params[2][0], transf.params[2][1], transf.params[2][2]) )
print ("# ]")
print ("#")
print ("#")
print ("# list of matched stars")
print ("#")
for i in range ( len (list_matched_1) ):
    print ("(%10.4f, %10.4f) on 1st image ==> (%10.4f, %10.4f) on 2nd image" \
           % (list_matched_1[i][0], list_matched_1[i][1], \
              list_matched_2[i][0], list_matched_2[i][1]) )

# reading FITS files
(header1, image1) = read_fits (file_fits1)
(header2, image2) = read_fits (file_fits2)

# marker and colour names for matplotlib
markers = ['o', 'v', '^', 's', 'p', 'h', '8']
colours = ['maroon', 'red', 'coral', 'bisque', 'orange', \
           'wheat', 'yellow', 'green', 'lime', 'aqua', \
           'skyblue', 'blue', 'indigo', 'violet', 'pink']

# making objects "fig" and "ax"
fig = matplotlib.figure.Figure ()
matplotlib.backends.backend_agg.FigureCanvasAgg (fig)
ax1 = fig.add_subplot (121)
ax2 = fig.add_subplot (122)

# plotting first image
norm1 \
    = astropy.visualization.mpl_normalize.ImageNormalize \
    ( stretch=astropy.visualization.HistEqStretch (image1) )
im1 = ax1.imshow (image1, origin='lower', cmap='bone', norm=norm1)
for i in range ( len (list_matched_1) ):
    i_marker = i % len (markers)
    i_colour = i % len (colours)
    ax1.plot (list_matched_1[i][0], list_matched_1[i][1], \
              marker=markers[i_marker], color=colours[i_colour], \
              markersize=8, fillstyle='none')
ax1.set_title ('First Image')

# plotting second image
norm2 \
    = astropy.visualization.mpl_normalize.ImageNormalize \
    ( stretch=astropy.visualization.HistEqStretch (image2) )
im2 = ax2.imshow (image2, origin='lower', cmap='bone', norm=norm2)
for i in range ( len (list_matched_2) ):
    i_marker = i % len (markers)
    i_colour = i % len (colours)
    ax2.plot (list_matched_2[i][0], list_matched_2[i][1], \
              marker=markers[i_marker], color=colours[i_colour], \
              markersize=8, fillstyle='none')
ax2.set_title ('Second Image')

# writing to a file
fig.savefig (file_fig, dpi=225)
```

Execute the script.

```
% chmod a+x ao2021_s19_06.py
% ./ao2021_s19_06.py -h
usage: ao2021_s19_06.py [-h] [-o FILE_OUTPUT]
                        catalogue1 catalogue2 fits1 fits2

finding star-to-star correspondence

positional arguments:
  catalogue1            catalogue file 1
  catalogue2            catalogue file 2
  fits1                 FITS file 1
  fits2                 FITS file 2

optional arguments:
  -h, --help            show this help message and exit
  -o FILE_OUTPUT, --file-output FILE_OUTPUT
                        output figure file

% ./ao2021_s19_06.py -o match_0.pdf stars_0.cat stars_1.cat \
? stars_0.fits stars_1.fits
#
# result of image alignment
#
#   date/time = 2021-05-27 20:42:31.730043
#
# input files
#
#   catalogue file 1 = stars_0.cat
#   catalogue file 2 = stars_1.cat
#
# transformation matrix
#
# [
#   [0.965925, 0.258817, -291.029791],
#   [-0.258817, 0.965925, 285.177948],
#   [0.000000, 0.000000, 1.000000]
# ]
#
#
# list of matched stars
#
(  427.6536,   718.7830) on 1st image ==> (  581.9708,   604.8288) on 2nd image
( 1754.4875,   533.9240) on 1st image ==> ( 1911.4250,   769.6463) on 2nd image
( 1144.3900,   387.9610) on 1st image ==> ( 1359.9034,   470.8140) on 2nd image
(  892.4160,   445.5007) on 1st image ==> ( 1101.6212,   461.1451) on 2nd image
( 1089.1050,  1265.9480) on 1st image ==> ( 1079.3278,  1304.7173) on 2nd image
( 1036.5090,   989.3659) on 1st image ==> ( 1100.0430,  1023.8028) on 2nd image
( 1449.8699,   608.5264) on 1st image ==> ( 1597.8592,   762.8984) on 2nd image
(  355.5129,   887.3405) on 1st image ==> (  468.6425,   748.9694) on 2nd image
(   91.5945,  1299.5430) on 1st image ==> (  107.0605,  1078.8181) on 2nd image
( 1891.6832,   710.5214) on 1st image ==> ( 1998.2563,   975.7711) on 2nd image
(  210.0494,   958.5712) on 1st image ==> (  309.7215,   780.1374) on 2nd image
( 1187.0417,  1280.8644) on 1st image ==> ( 1170.1186,  1344.1679) on 2nd image
( 1789.5059,  1330.9291) on 1st image ==> ( 1738.9862,  1548.6088) on 2nd image
(  522.5328,   726.6528) on 1st image ==> (  671.5495,   636.9774) on 2nd image
( 1389.2390,  1453.7359) on 1st image ==> ( 1320.5294,  1563.6546) on 2nd image
( 1759.6721,   996.8308) on 1st image ==> ( 1796.6302,  1218.1563) on 2nd image
( 1832.8962,   391.6778) on 1st image ==> ( 2024.0220,   652.5956) on 2nd image
(   26.9681,   928.8527) on 1st image ==> (  140.5583,   704.0075) on 2nd image
```

```
( 1251.5349,   712.6292) on 1st image ==> ( 1379.3614,   812.1382) on 2nd image
(  950.9102,  1186.2260) on 1st image ==> (  966.4180,  1191.7817) on 2nd image
( 1614.0986,  1031.0432) on 1st image ==> ( 1647.1573,  1213.5161) on 2nd image
( 1008.8435,   103.1927) on 1st image ==> ( 1302.7021,   160.6272) on 2nd image
(  930.5438,   125.0077) on 1st image ==> ( 1221.4373,   161.4614) on 2nd image
(  716.4430,   203.5878) on 1st image ==> (  994.2842,   181.9682) on 2nd image
( 1220.9639,  1724.3945) on 1st image ==> ( 1087.9755,  1781.5115) on 2nd image
( 1818.8588,   808.2168) on 1st image ==> ( 1902.6105,  1051.2960) on 2nd image
% ls -l match_0.pdf
-rw-r--r--  1 daisuke  taiwan  1194508 May 27 20:42 match_0.pdf
```

Check the PDF file. (Fig. 6)

```
% xpdf match_0.pdf
```

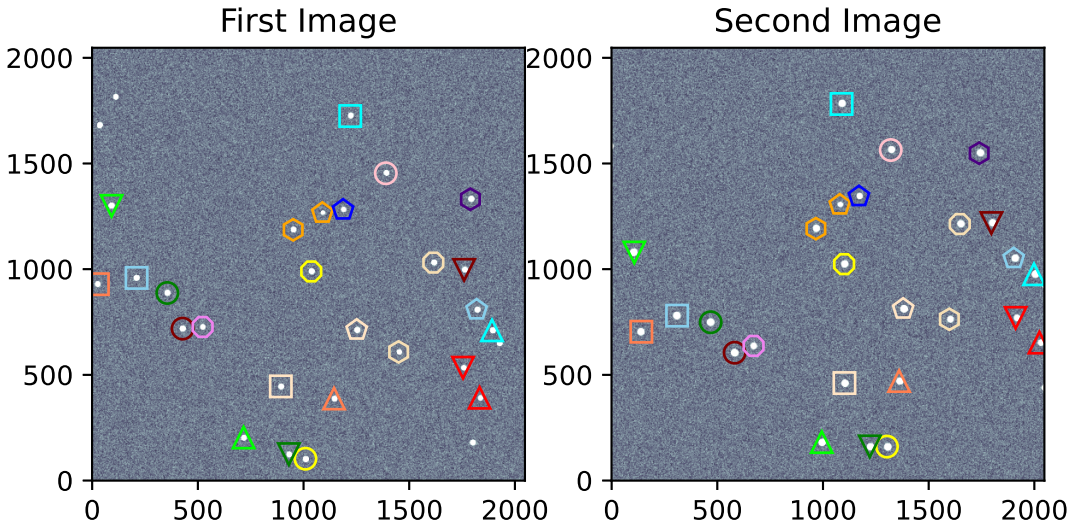

Figure 6: Result of finding star-to-star correspondence.

8 Aligining the image

Make a Python script to align an image to the reference image.

Python Code 7: ao2021_s19_07.py

```
#!/usr/pkg/bin/python3.9
```

```
# importing argparse module
import argparse

# importing sys module
import sys

# importing datetime module
import datetime

# importing numpy module
import numpy

# importing astropy module
import astropy.table
import astropy.visualization

# importing scikit-image module
import skimage.transform

# importing astroalign module
import astroalign

# importing matplotlib module
import matplotlib.figure
import matplotlib.backends.backend_agg

# date/time
now = datetime.datetime.now ()

# constructing parser object
desc   = 'aligning image'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-o', '--file-output', default='', \
                     help='output figure file')
parser.add_argument ('catalogue1', nargs=1, help='catalogue file 1')
parser.add_argument ('catalogue2', nargs=1, help='catalogue file 2')
parser.add_argument ('fits1', nargs=1, help='FITS file 1')
parser.add_argument ('fits2', nargs=1, help='FITS file 2')

# command-line argument analysis
args = parser.parse_args ()

# file names
file_cat1  = args.catalogue1[0]
file_cat2  = args.catalogue2[0]
file_fits1 = args.fits1[0]
file_fits2 = args.fits2[0]
file_fig   = args.file_output

# check of output file name
if not ( (file_fig[-4:] == '.eps') or (file_fig[-4:] == '.pdf') \
         or (file_fig[-4:] == '.png') or (file_fig[-3:] == '.ps') ):
    print ("Figure file name must be either EPS, PDF, PNG, or PS.")
    sys.exit ()

# check of catalogue file name
```

```
if not ( (file_cat1[-4:] == '.cat') and (file_cat2[-4:] == '.cat') ):
    print ("Input file must be a catalogue file (*.cat).")
    print ("catalogue file 1 = %s" % file_cat1)
    print ("catalogue file 2 = %s" % file_cat2)
    sys.exit ()

# check of FITS file name
if not ( (file_fits1[-5:] == '.fits') and (file_fits2[-5:] == '.fits') ):
    print ("Input file must be a FITS file (*.fits).")
    print ("FITS file 1 = %s" % file_fits1)
    print ("FITS file 2 = %s" % file_fits2)
    sys.exit ()

# function to read a FITS file
def read_fits (file_fits):
    # reading FITS file
    with astropy.io.fits.open (file_fits) as hdu:
        # reading header and image
        header = hdu[0].header
        image  = hdu[0].data
        # if no image in PrimaryHDU, then read next HDU
        if (header['NAXIS'] == 0):
            header = hdu[1].header
            image  = hdu[1].data
    # returning header and image
    return (header, image)

# reading catalogue from a file
table_source1 = astropy.table.Table.read (file_cat1, \
                                          format='ascii.commented_header')
table_source2 = astropy.table.Table.read (file_cat2, \
                                          format='ascii.commented_header')

# (x, y) coordinates of sources
list_source1_x = list (table_source1['xcentroid'])
list_source1_y = list (table_source1['ycentroid'])
list_source2_x = list (table_source2['xcentroid'])
list_source2_y = list (table_source2['ycentroid'])
position_1 = numpy.transpose ( (list_source1_x, list_source1_y) )
position_2 = numpy.transpose ( (list_source2_x, list_source2_y) )

# finding star-to-star matching
transf, (list_matched_2, list_matched_1) \
    = astroalign.find_transform (position_2, position_1)

# transformation
list_matched_2_aligned \
    = astroalign.matrix_transform (list_matched_2, transf.params)

# printing results
print ("#")
print ("# result of image alignment")
print ("#")
print ("#   date/time = %s" % now)
print ("#")
print ("# input files")
print ("#")
print ("#   catalogue file 1 = %s" % file_cat1)
print ("#   catalogue file 2 = %s" % file_cat2)
```

```
print ("#")
print ("# transformation matrix")
print ("#")
print ("# [")
print ("#   [%f, %f, %f]," \
       % (transf.params[0][0], transf.params[0][1], transf.params[0][2]) )
print ("#   [%f, %f, %f]," \
       % (transf.params[1][0], transf.params[1][1], transf.params[1][2]) )
print ("#   [%f, %f, %f]" \
       % (transf.params[2][0], transf.params[2][1], transf.params[2][2]) )
print ("# ]")
print ("#")
print ("#")
print ("# list of matched stars")
print ("#")
for i in range ( len (list_matched_1) ):
    print ("(%10.4f, %10.4f) on 1st image ==> (%10.4f, %10.4f) on 2nd image" \
           % (list_matched_1[i][0], list_matched_1[i][1], \
              list_matched_2[i][0], list_matched_2[i][1]) )

# reading FITS files
(header1, image1) = read_fits (file_fits1)
(header2, image2) = read_fits (file_fits2)

# byte swap
# data stored in FITS file is network byte-order (big endian).
# Intel/AMD CPUs use little endian.
image1 = image1.byteswap ().newbyteorder ()
image2 = image2.byteswap ().newbyteorder ()

# aligning 2nd image to 1st image
st = skimage.transform.SimilarityTransform (scale=transf.scale, \
                                            rotation=transf.rotation, \
                                            translation=transf.translation)
image2_aligned = skimage.transform.warp (image2, st.inverse)

# marker and colour names for matplotlib
markers = ['o', 'v', '^', 's', 'p', 'h', '8']
colours = ['maroon', 'red', 'coral', 'bisque', 'orange', \
           'wheat', 'yellow', 'green', 'lime', 'aqua', \
           'skyblue', 'blue', 'indigo', 'violet', 'pink']

# making objects "fig" and "ax"
fig = matplotlib.figure.Figure ()
matplotlib.backends.backend_agg.FigureCanvasAgg (fig)
ax1 = fig.add_subplot (121)
ax2 = fig.add_subplot (122)

# plotting first image
norm1 \
    = astropy.visualization.mpl_normalize.ImageNormalize \
    ( stretch=astropy.visualization.HistEqStretch (image1) )
im1 = ax1.imshow (image1, origin='lower', cmap='bone', norm=norm1)
for i in range ( len (list_matched_1) ):
    i_marker = i % len (markers)
    i_colour = i % len (colours)
    ax1.plot (list_matched_1[i][0], list_matched_1[i][1], \
              marker=markers[i_marker], color=colours[i_colour], \
              markersize=8, fillstyle='none')
```

```
ax1.set_title ('First Image')

# plotting second image
norm2 \
    = astropy.visualization.mpl_normalize.ImageNormalize \
    ( stretch=astropy.visualization.HistEqStretch (image2_aligned) )
im2 = ax2.imshow (image2_aligned, origin='lower', cmap='bone', norm=norm2)
for i in range ( len (list_matched_2_aligned) ):
    i_marker = i % len (markers)
    i_colour = i % len (colours)
    ax2.plot (list_matched_2_aligned[i][0], list_matched_2_aligned[i][1], \
              marker=markers[i_marker], color=colours[i_colour], \
              markersize=8, fillstyle='none')
ax2.set_title ('Second Image')

# writing to a file
fig.savefig (file_fig, dpi=225)
```

Run the script.

```
% chmod a+x ao2021_s19_07.py
% ./ao2021_s19_07.py -h
usage: ao2021_s19_07.py [-h] [-o FILE_OUTPUT]
                        catalogue1 catalogue2 fits1 fits2

aligning image

positional arguments:
  catalogue1            catalogue file 1
  catalogue2            catalogue file 2
  fits1                 FITS file 1
  fits2                 FITS file 2

optional arguments:
  -h, --help            show this help message and exit
  -o FILE_OUTPUT, --file-output FILE_OUTPUT
                        output figure file

% ./ao2021_s19_07.py -o match_1.pdf stars_0.cat stars_1.cat \
? stars_0.fits stars_1.fits
#
# result of image alignment
#
#   date/time = 2021-05-27 23:51:53.865153
#
# input files
#
#   catalogue file 1 = stars_0.cat
#   catalogue file 2 = stars_1.cat
#
# transformation matrix
#
# [
#   [0.965925, 0.258817, -291.029791],
#   [-0.258817, 0.965925, 285.177948],
#   [0.000000, 0.000000, 1.000000]
# ]
#
```

```
#
# list of matched stars
#
(  427.6536,   718.7830) on 1st image ==> (  581.9708,   604.8288) on 2nd image
( 1754.4875,   533.9240) on 1st image ==> ( 1911.4250,   769.6463) on 2nd image
( 1144.3900,   387.9610) on 1st image ==> ( 1359.9034,   470.8140) on 2nd image
(  892.4160,   445.5007) on 1st image ==> ( 1101.6212,   461.1451) on 2nd image
( 1089.1050,  1265.9480) on 1st image ==> ( 1079.3278,  1304.7173) on 2nd image
( 1036.5090,   989.3659) on 1st image ==> ( 1100.0430,  1023.8028) on 2nd image
( 1449.8699,   608.5264) on 1st image ==> ( 1597.8592,   762.8984) on 2nd image
(  355.5129,   887.3405) on 1st image ==> (  468.6425,   748.9694) on 2nd image
(   91.5945,  1299.5430) on 1st image ==> (  107.0605,  1078.8181) on 2nd image
( 1891.6832,   710.5214) on 1st image ==> ( 1998.2563,   975.7711) on 2nd image
(  210.0494,   958.5712) on 1st image ==> (  309.7215,   780.1374) on 2nd image
( 1187.0417,  1280.8644) on 1st image ==> ( 1170.1186,  1344.1679) on 2nd image
( 1789.5059,  1330.9291) on 1st image ==> ( 1738.9862,  1548.6088) on 2nd image
(  522.5328,   726.6528) on 1st image ==> (  671.5495,   636.9774) on 2nd image
( 1389.2390,  1453.7359) on 1st image ==> ( 1320.5294,  1563.6546) on 2nd image
( 1759.6721,   996.8308) on 1st image ==> ( 1796.6302,  1218.1563) on 2nd image
( 1832.8962,   391.6778) on 1st image ==> ( 2024.0220,   652.5956) on 2nd image
(   26.9681,   928.8527) on 1st image ==> (  140.5583,   704.0075) on 2nd image
( 1251.5349,   712.6292) on 1st image ==> ( 1379.3614,   812.1382) on 2nd image
(  950.9102,  1186.2260) on 1st image ==> (  966.4180,  1191.7817) on 2nd image
( 1614.0986,  1031.0432) on 1st image ==> ( 1647.1573,  1213.5161) on 2nd image
( 1008.8435,   103.1927) on 1st image ==> ( 1302.7021,   160.6272) on 2nd image
(  930.5438,   125.0077) on 1st image ==> ( 1221.4373,   161.4614) on 2nd image
(  716.4430,   203.5878) on 1st image ==> (  994.2842,   181.9682) on 2nd image
( 1220.9639,  1724.3945) on 1st image ==> ( 1087.9755,  1781.5115) on 2nd image
( 1818.8588,   808.2168) on 1st image ==> ( 1902.6105,  1051.2960) on 2nd image
% ls -l match_1.pdf
-rw-r--r--  1 daisuke  taiwan  1192497 May 27 23:51 match_1.pdf
```

Display the PDF file, and check the result. (Fig. 7)

```
% xpdf match_1.pdf
```

9 Dealing with real data

Next, we deal with real data.

9.1 Downloading SDSS image

Make a Python script to download DSS/SDSS images.

Python Code 8: ao2021_s19_08.py

```
#!/usr/pkg/bin/python3.9

# importing argparse module
import argparse

# importing sys module
import sys

# importing astroquery module
import astroquery.simbad
```

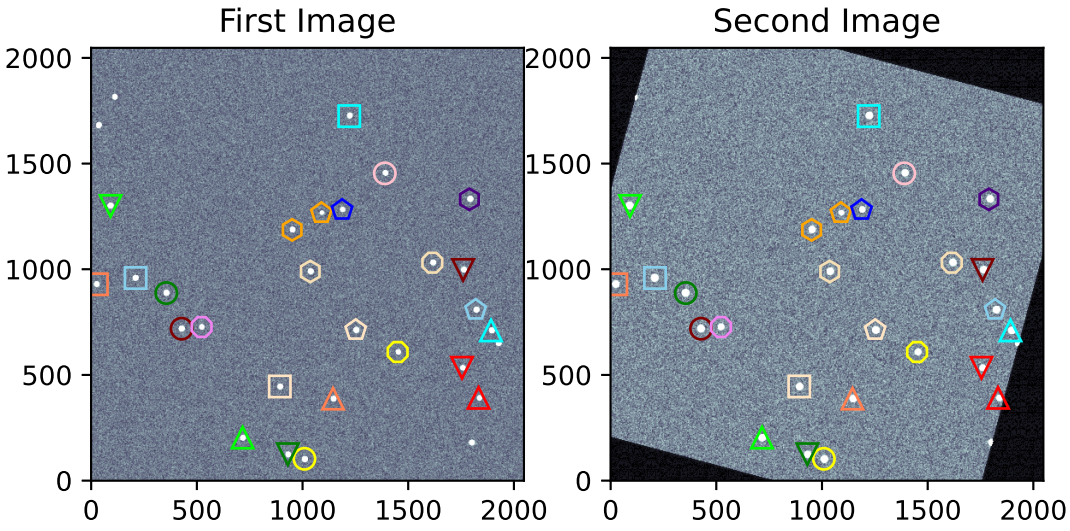

Figure 7: Result of finding star-to-star correspondence. The 2nd image is aligned to the 1st image.

```
import astroquery.ned
import astroquery.skyview

# importing astropy module
import astropy.coordinates
import astropy.units

# importing datetime module
import datetime

# importing ssl module
import ssl

# allow insecure downloading
ssl._create_default_https_context = ssl._create_unverified_context

# date/time
now = datetime.datetime.now ().isoformat ()

# units
u_ha  = astropy.units.hourangle
u_deg = astropy.units.deg

# constructing parser object
desc = "downloading DSS/SDSS image"
parser = argparse.ArgumentParser (description=desc)

# adding arguments
list_resolver = ['simbad', 'ned']
list_survey   = ['DSS1 Blue', 'DSS1 Red', 'DSS2 Blue', 'DSS2 Red', 'DSS2 IR', \
                 'SDSSu', 'SDSSg', 'SDSSr', 'SDSSi', 'SDSSz']
parser.add_argument ('-r', '--resolver', choices=list_resolver, \
                     default='simbad', help='choice of name resolver')
parser.add_argument ('-s', '--survey', choices=list_survey, \
                     default='SDSSr', help='choice of survey')
parser.add_argument ('-t', '--target', default='', help='target name')
parser.add_argument ('-f', '--fov', type=int, default=1024, \
                     help='field-of-view in pixel')
parser.add_argument ('-o', '--output', default='', help='output file name')

# command-line argument analysis
args = parser.parse_args ()

# input parameters
name_resolver = args.resolver
survey        = args.survey
target_name   = args.target
fov_pix       = args.fov
file_output   = args.output

# checking target name
if (target_name == ''):
    # printing error message
    print ("No target name is given!")
    # exit
    sys.exit ()

# checking output file name
if (file_output == ''):
```

```
    # printing error message
    print ("No output file name is given!")
    # exit
    sys.exit ()
elif not (file_output[-5:] == '.fits'):
    # printing error message
    print ("Output file must be FITS file!")
    # exit
    sys.exit ()

# using name resolver
if (name_resolver == 'simbad'):
    query_result = astroquery.simbad.Simbad.query_object (target_name)
elif (name_resolver == 'ned'):
    query_result = astroquery.ned.Ned.query_object (target_name)

# RA and Dec
RA  = query_result['RA']
Dec = query_result['DEC']

# coordinate
if (name_resolver == 'simbad'):
    coord = astropy.coordinates.SkyCoord (RA[0], Dec[0], unit=(u_ha, u_deg))
elif (name_resolver == 'ned'):
    coord = astropy.coordinates.SkyCoord (RA[0], Dec[0], unit=(u_deg, u_deg))

coord_str = coord.to_string (style='hmsdms')
(coord_ra_str, coord_dec_str) = coord_str.split ()
coord_ra_deg  = coord.ra.deg
coord_dec_deg = coord.dec.deg

# printing coordinate
print ("Target Name: %s" % target_name)
print ("  RA:  %s = %f deg" % (coord_ra_str, coord_ra_deg) )
print ("  Dec: %s = %f deg" % (coord_dec_str, coord_dec_deg) )

# searching image
list_image = astroquery.skyview.SkyView.get_image_list (position=coord, \
                                                         survey=survey)

# printing image list
print ("Available images:")
print (" ", list_image)

# getting image
image = astroquery.skyview.SkyView.get_images (position=coord, survey=survey, \
                                               pixels=fov_pix)

# header and data
image0 = image[0]
header = image0[0].header
data   = image0[0].data

# adding comments in header
header['history'] = "image downloaded from %s" % survey
header['history'] = "image saved on %s" % now

# saving to a FITS file
astropy.io.fits.writeto (file_output, data, header=header)
```

Execute the script, and download SDSS image.

```
% chmod a+x ao2021_s19_08.py
% ./ao2021_s19_08.py -h
usage: ao2021_s19_08.py [-h] [-r {simbad,ned}]
                        [-s {DSS1 Blue,DSS1 Red,DSS2 Blue,DSS2 Red,DSS2 IR,SDSSu
,SDSSg,SDSSr,SDSSi,SDSSz}]
                        [-t TARGET] [-f FOV] [-o OUTPUT]

downloading DSS/SDSS image

optional arguments:
  -h, --help            show this help message and exit
  -r {simbad,ned}, --resolver {simbad,ned}
                        choice of name resolver
  -s {DSS1 Blue,DSS1 Red,DSS2 Blue,DSS2 Red,DSS2 IR,SDSSu,SDSSg,SDSSr,SDSSi,SDSS
z}, --survey {DSS1 Blue,DSS1 Red,DSS2 Blue,DSS2 Red,DSS2 IR,SDSSu,SDSSg,SDSSr,SD
SSi,SDSSz}
                        choice of survey
  -t TARGET, --target TARGET
                        target name
  -f FOV, --fov FOV     field-of-view in pixel
  -o OUTPUT, --output OUTPUT
                        output file name

% ./ao2021_s19_08.py -s SDSSi -t PG1047+003 -f 2048 -o pg1047_sdss_i.fits
Target Name: PG1047+003
  RA:  10h50m02.8264s = 162.511777 deg
  Dec: -00d00m36.88s = -0.010244 deg
Available images:
  ['https://skyview.gsfc.nasa.gov/tempspace/fits/skv20222354931136.fits']
Downloading https://skyview.gsfc.nasa.gov/tempspace/fits/skv20222436354319.fits
|=========================================|  16M/ 16M (100.00%)        36s
% ls -l pg1047_sdss_i.fits
-rw-r--r--  1 daisuke  taiwan  16793280 May 28 00:10 pg1047_sdss_i.fits
```

Use Ginga to check the image. (Fig. 8)

```
% ginga pg1047_sdss_i.fits &
```

9.2 Downloading DSS image

Download DSS image.

```
% ./ao2021_s19_08.py -s "DSS2 Blue" -t PG1047+003 -f 768 -o pg1047_dss_b.fits
Target Name: PG1047+003
  RA:  10h50m02.8264s = 162.511777 deg
  Dec: -00d00m36.88s = -0.010244 deg
Available images:
  ['https://skyview.gsfc.nasa.gov/tempspace/fits/skv20223128125760.fits']
Downloading https://skyview.gsfc.nasa.gov/tempspace/fits/skv20223845156098.fits
|=========================================| 2.3M/2.3M (100.00%)         5s
% ls -l pg1047_*
-rw-r--r--  1 daisuke  taiwan   2373120 May 28 00:19 pg1047_dss_b.fits
-rw-r--r--  1 daisuke  taiwan  16793280 May 28 00:10 pg1047_sdss_i.fits
```

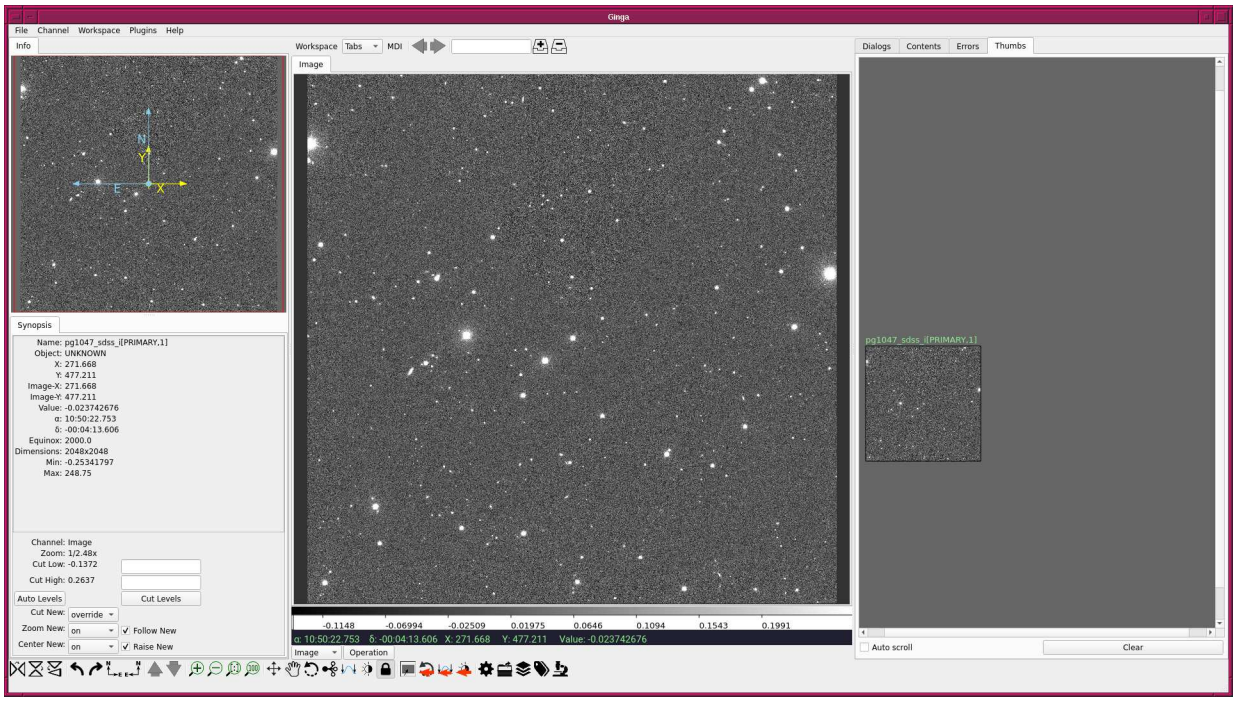

Figure 8: SDSS i'-band image of PG1047+003.

Use Ginga to check the image. (Fig. 9)

```
% ginga pg1047_dss_b.fits &
```

9.3 Source extraction

Carry out source extraction for both SDSS and DSS images.

```
% ./ao2021_s19_04.py -i pg1047_sdss_i.fits -t 100.0 -o pg1047_sdss_i.cat \
? -g pg1047_sdss_i.pdf
% ./ao2021_s19_04.py -i pg1047_dss_b.fits -t 30.0 -o pg1047_dss_b.cat \
? -g pg1047_dss_b.pdf
% ls -l pg1047_*.cat
-rw-r--r--  1 daisuke  taiwan   7050 May 28 00:33 pg1047_dss_b.cat
-rw-r--r--  1 daisuke  taiwan  11232 May 28 00:33 pg1047_sdss_i.cat
```

Results of source extraction are shown in Fig. 10 and 11.

9.4 Image alignment

Carry out star-to-star matching.

```
% ./ao2021_s19_06.py -o match_2.pdf pg1047_sdss_i.cat pg1047_dss_b.cat \
? pg1047_sdss_i.fits pg1047_dss_b.fits
#
# result of image alignment
#
#   date/time = 2021-05-28 00:39:50.721716
#
# input files
```

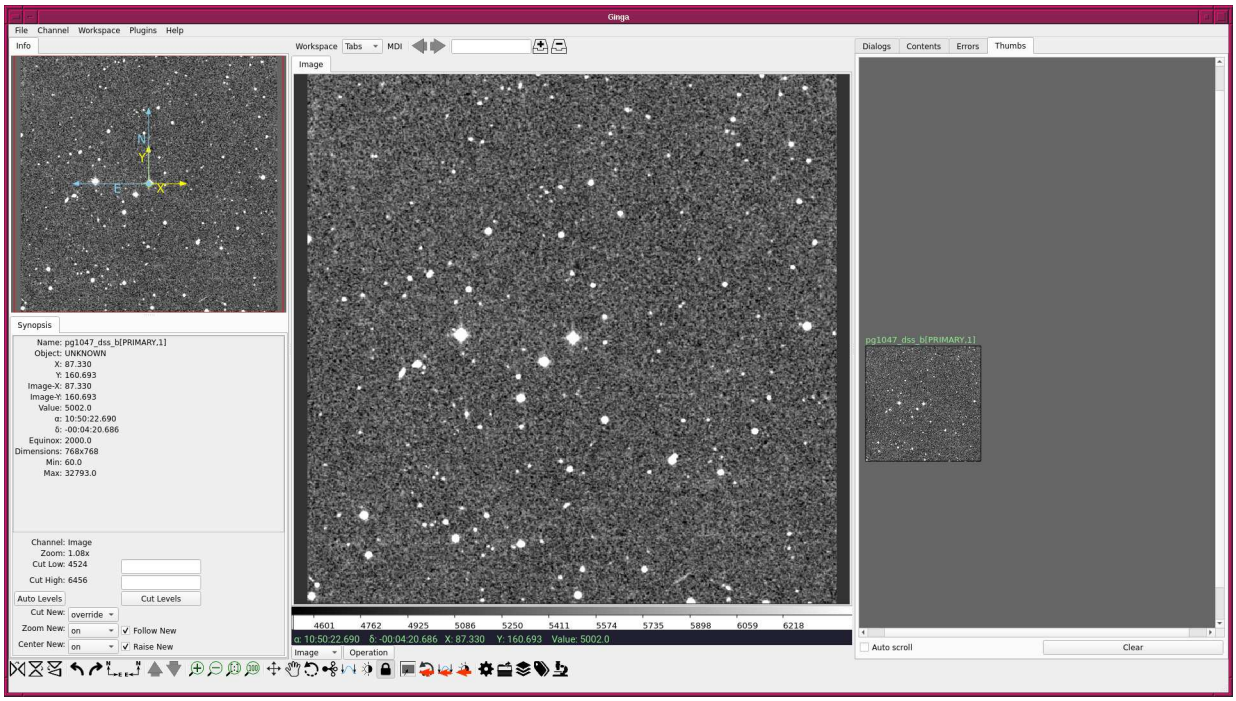

Figure 9: DSS Blue image of PG1047+003.

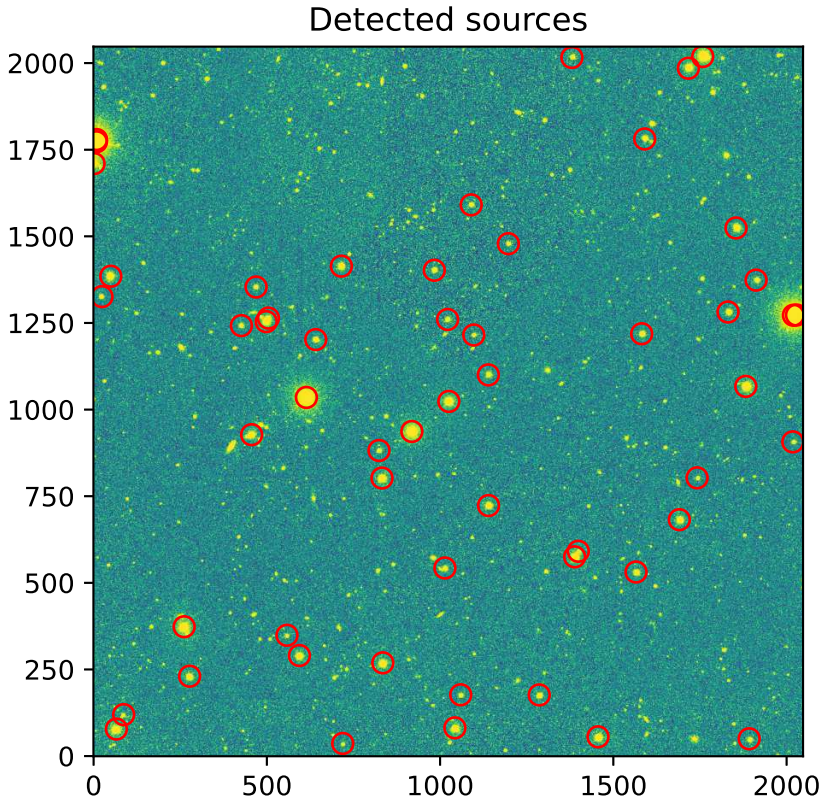

Figure 10: Result of source extraction of SDSS image.

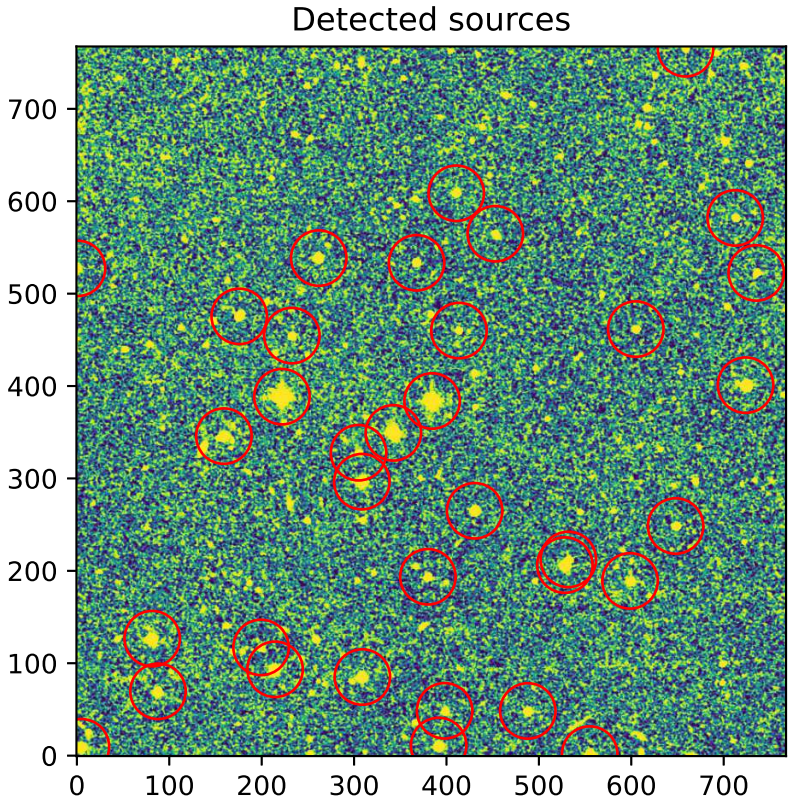

Figure 11: Result of source extraction of DSS image.

```
#
#   catalogue file 1 = pg1047_sdss_i.cat
#   catalogue file 2 = pg1047_dss_b.cat
#
# transformation matrix
#
# [
#   [2.525859, -0.000424, 53.972876],
#   [0.000424, 2.525859, 53.839742],
#   [0.000000, 0.000000, 1.000000]
# ]
#
#
# list of matched stars
#
(  823.6800,   882.0045) on 1st image ==> (  304.8340,   327.8275) on 2nd image
(  498.3468,  1253.8589) on 1st image ==> (  175.7000,   475.3944) on 2nd image
( 1059.4451,   176.9533) on 1st image ==> (  398.0028,    48.5176) on 2nd image
( 1398.8033,   591.1109) on 1st image ==> (  531.8952,   212.3450) on 2nd image
( 1286.1954,   176.4987) on 1st image ==> (  487.9928,    48.3692) on 2nd image
(  983.2979,  1402.0423) on 1st image ==> (  367.6290,   533.3973) on 2nd image
(  832.3487,   802.4233) on 1st image ==> (  308.3825,   296.5177) on 2nd image
( 1581.7806,  1218.8741) on 1st image ==> (  604.8763,   461.6491) on 2nd image
(  918.5140,   936.9002) on 1st image ==> (  342.5177,   349.3132) on 2nd image
( 1013.9307,   542.8151) on 1st image ==> (  379.8766,   193.5695) on 2nd image
(   65.7829,    77.9935) on 1st image ==> (    4.9078,     9.6261) on 2nd image
(  715.1062,  1413.9673) on 1st image ==> (  261.7456,   538.4762) on 2nd image
(  594.0550,   290.2665) on 1st image ==> (  214.5051,    93.4324) on 2nd image
( 1025.2500,  1023.6115) on 1st image ==> (  384.4590,   383.7996) on 2nd image
(  558.1739,   348.6601) on 1st image ==> (  199.3383,   117.1204) on 2nd image
( 1042.7106,    81.4663) on 1st image ==> (  391.4883,    11.0018) on 2nd image
(  614.1969,  1035.1081) on 1st image ==> (  221.9186,   388.4949) on 2nd image
( 1854.1293,  1524.1565) on 1st image ==> (  712.4975,   581.9372) on 2nd image
( 1911.9819,  1373.6673) on 1st image ==> (  735.2345,   522.4108) on 2nd image
( 1690.5344,   682.2595) on 1st image ==> (  647.9930,   248.5211) on 2nd image
(  641.1427,  1202.0095) on 1st image ==> (  232.6394,   454.6186) on 2nd image
( 1882.3096,  1066.9590) on 1st image ==> (  723.6902,   400.9431) on 2nd image
(  277.2246,   230.5123) on 1st image ==> (   87.7854,    69.6134) on 2nd image
( 1565.4674,   531.2258) on 1st image ==> (  598.6880,   189.1224) on 2nd image
( 1388.3806,   575.0339) on 1st image ==> (  528.1273,   206.2133) on 2nd image
( 1139.8708,   723.0177) on 1st image ==> (  430.4903,   265.0167) on 2nd image
(  834.7360,   269.1596) on 1st image ==> (  308.9729,    85.0933) on 2nd image
(  456.1203,   927.5883) on 1st image ==> (  159.1240,   345.7897) on 2nd image
% ls -l match_2.pdf
-rw-r--r--  1 daisuke  taiwan  1340520 May 28 00:39 match_2.pdf
```

Display the PDF file. (Fig. 12)

```
% xpdf match_2.pdf
```

10 For your training

Read followings.

1. "Data Tables" of Astropy
    - https://docs.astropy.org/en/stable/table/

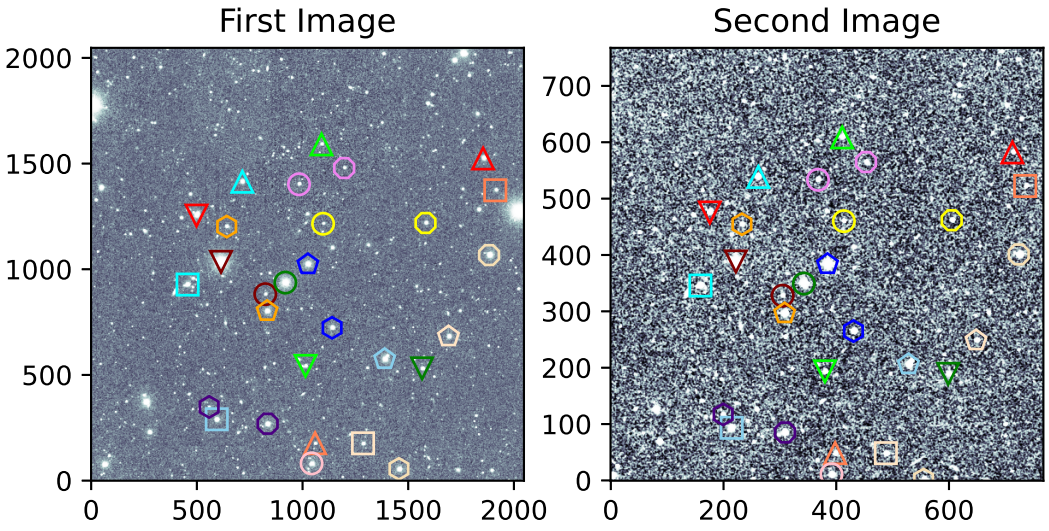

Figure 12: The star-to-star correspondence between SDSS and DSS images.

2. “FITS File Handling” of Astropy
    - `https://docs.astropy.org/en/stable/io/fits/`
3. “Visualization” of Astropy
    - `https://docs.astropy.org/en/stable/visualization/`
4. “Random sampling” of Numpy
    - `https://numpy.org/doc/stable/reference/random/`
5. “Datasets” of photutils
    - `https://photutils.readthedocs.io/en/stable/datasets.html`
6. “Image Segmentation” of photutils
    - `https://photutils.readthedocs.io/en/stable/segmentation.html`
7. “astroalign”
    - `https://astroalign.readthedocs.io/`
8. “scikit-image”
    - `https://scikit-image.org/`
9. “astroquery”
    - `https://astroquery.readthedocs.io/`

11 Assignment

1. Describe mathematical expression of translation of 2-dimensional coordinates (x, y).
2. Describe mathematical expression of rotation of 2-dimensional coordinates (x, y).
3. Choose one r’-band and i’-band image of SDSS images. Make your own Python script to try image alignment. Show the source code of your Python script. Show the result of your star-to-star correspondence and image alignment. What is the transformation matrix?
4. Choose one g’-band image of Lulin One-meter Telescope data. Choose one r’-band image of Lulin One-meter Telescoep data of the same field as g’-band data. Make your own Python script to try image alignment. Show the source code of your Python script. Show the result of your star-to-star correspondence and image alignment. What is the transformation matrix?
5. Choose one g’-band image of Lulin One-meter Telescope data. Download SDSS image of the same field. Make your own Python script to try image alignment. Show the source code of your Python script. Show the result of your star-to-star correspondence and image alignment. What is the transformation matrix?