Advanced Astronomical Observations 2021
Session 23: World Coordinate System (WCS) 2

Kinoshita Daisuke

11 June 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 examine WCS-related keywords in FITS files.

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: 320, done.
remote: Counting objects: 100% (118/118), done.
remote: Compressing objects: 100% (113/113), done.
Receiving objects: 100% (320/320), 5.42 MiB | 1.69 MiB/s, done.
remote: Total 320 (delta 55), reused 0 (delta 0), pack-reused 202
Resolving deltas: 100% (163/163), done.
% ls
ncu_advobs_202102/
% ls -l ncu_advobs_202102/
total 1
-rw-r--r--  1 daisuke  wheel  35149 Jun 10 23:08 LICENSE
-rw-r--r--  1 daisuke  wheel    568 Jun 10 23:08 README
-rw-r--r--  1 daisuke  wheel    687 Jun 10 23:08 REQUIREMENTS
-rw-r--r--  1 daisuke  wheel    342 Jun 10 23:08 download_python.csh
-rw-r--r--  1 daisuke  wheel    495 Jun 10 23:08 download_stds.csh
drwxr-xr-x  2 daisuke  wheel    512 Jun 10 23:08 s16_skybackground/
```

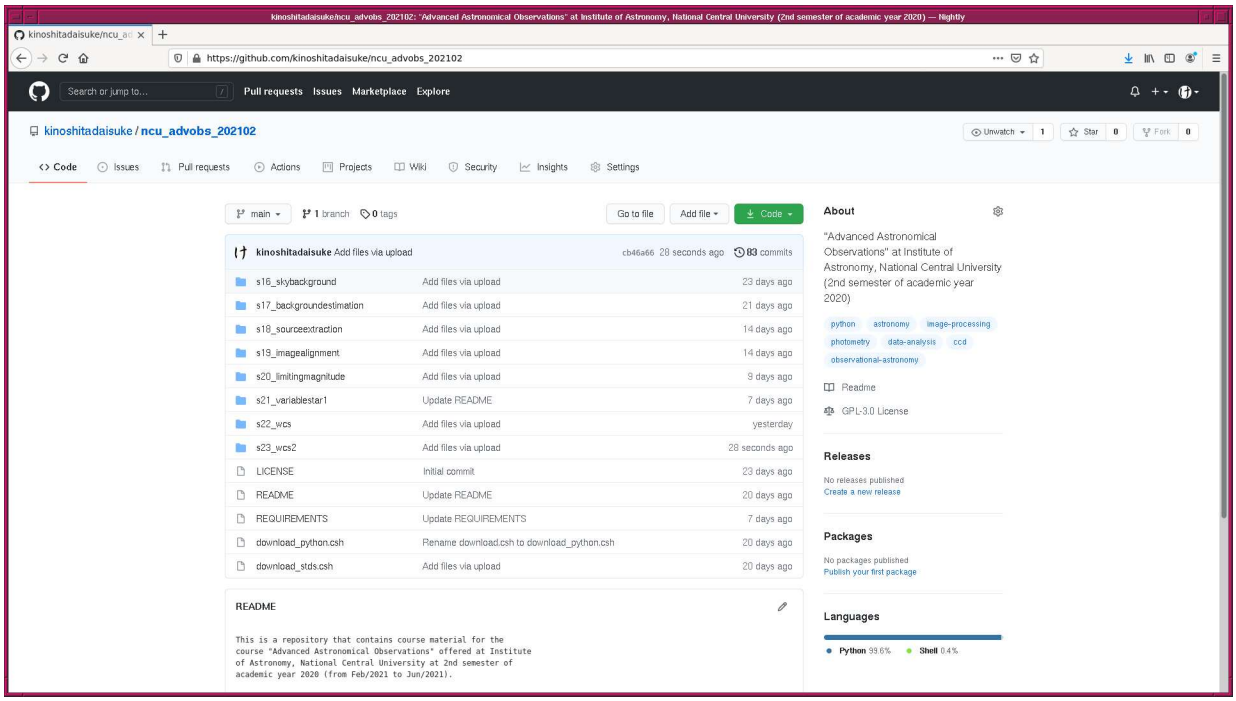

Figure 1: The GitHub repository for Python scripts for this course.

```
drwxr-xr-x  2 daisuke  wheel    512 Jun 10 23:08 s17_backgroundestimation/
drwxr-xr-x  2 daisuke  wheel    512 Jun 10 23:08 s18_sourceextraction/
drwxr-xr-x  2 daisuke  wheel    512 Jun 10 23:08 s19_imagealignment/
drwxr-xr-x  2 daisuke  wheel    512 Jun 10 23:08 s20_limitingmagnitude/
drwxr-xr-x  2 daisuke  wheel    512 Jun 10 23:08 s21_variablestar1/
drwxr-xr-x  2 daisuke  wheel    512 Jun 10 23:08 s22_wcs/
drwxr-xr-x  2 daisuke  wheel    512 Jun 10 23:08 s23_wcs2/
% ls -l ncu_advobs_202102/s23_wcs2/
total 1
-rw-r--r--  1 daisuke  wheel     62 Jun 10 23:08 README
-rw-r--r--  1 daisuke  wheel   3464 Jun 10 23:08 ao2021_s23_01.py
-rw-r--r--  1 daisuke  wheel  18981 Jun 10 23:08 ao2021_s23_02.py
-rw-r--r--  1 daisuke  wheel   4493 Jun 10 23:08 ao2021_s23_03.py
-rw-r--r--  1 daisuke  wheel   2082 Jun 10 23:08 ao2021_s23_04.py
% head -15 ncu_advobs_202102/s23_wcs2/ao2021_s23_01.py
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2021/06/10 20:18:25 (CST) daisuke>
#

# importing argparse module
import argparse

# importing sys module
import sys

# importing pathlib module
import pathlib
```

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 Copying FITS files

Make a Python script to copy FITS files.

Python Code 1: ao2021_s23_01.py

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

# importing argparse module
import argparse

# importing sys module
import sys

# importing pathlib module
import pathlib

# importing shutil module
import shutil

# importing astropy module
import astropy.io.fits

# list of data type
list_datatype = ['LIGHT', 'FLAT', 'DARK', 'BIAS']

# constructing parser object
desc = "copying FITS files"
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-t', '--data-type', choices=list_datatype, \
                     default='LIGHT', help='data type (default: LIGHT)')
parser.add_argument ('-e', '--exptime', type=float, default=0.0, \
                     help='exposure time in sec')
parser.add_argument ('-f', '--filter-name', default='', \
                     help='filter name')
parser.add_argument ('-o', '--object-name', default='', \
                     help='object name')
parser.add_argument ('-d', '--dir-dst', default='.', \
                     help='destination directory (default: .)')
parser.add_argument ('files', nargs='+', help='FITS files')

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

```
# input parameters
data_type   = args.data_type
exptime     = args.exptime
filter_name = args.filter_name
object_name = args.object_name
dir_dst     = args.dir_dst
files_fits  = args.files

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

# check of object name
if (object_name == ''):
    print ("Object name has to be specified.")
    sys.exit ()

# check of filter name
if (filter_name == ''):
    print ("Filter name has to be specified.")
    sys.exit ()

# destination directory
path_dst = pathlib.Path (dir_dst)
path_dst.mkdir (mode=0o755, exist_ok=True)

# processing FITS files one-by-one
for file_fits in files_fits:
    # check of file name
    if not (file_fits[-5:] == '.fits'):
        # printing a message to stderr
        print ("Input file \"%s\" is not a FITS file." % file_fits, \
               file=sys.stderr)
        print ("Input file must be \"*.fits\".")
        # skipping
        continue

    # making pathlib object
    path_fits = pathlib.Path (file_fits)
    filename  = path_fits.name

    # if the file does not exist, then skip
    if not ( path_fits.exists () and path_fits.is_file () ):
        # printing a message to stderr
        print ("The file \"%s\" does not exist or is not a regular file." \
               % file_fits, file=sys.stderr)
        # skipping
        continue

    # reading FITS header
    header = read_fits_header (file_fits)
```

```
    # data type
    if not (data_type == header['IMAGETYP']):
        continue

    # exposure time
    if not (exptime == header['EXPTIME']):
        continue

    # filter name
    if ('FILTER' in header):
        if not (filter_name == header['FILTER']):
            continue

    # object name
    if not (object_name == header['OBJECT']):
        continue

    # printing status
    print ("Now copying the file \"%s\" to \"%s\"..." % (file_fits, dir_dst) )

    # copying file
    shutil.copy2 (path_fits, path_dst)
```

Execute the script, and copy r'-band 60-sec data of V0678 Vir.

```
% chmod a+x ao2021_s23_01.py
% ./ao2021_s23_01.py -h
usage: ao2021_s23_01.py [-h] [-t {LIGHT,FLAT,DARK,BIAS}] [-e EXPTIME]
                        [-f FILTER_NAME] [-o OBJECT_NAME] [-d DIR_DST]
                        files [files ...]

copying FITS files

positional arguments:
  files                 FITS files

optional arguments:
  -h, --help            show this help message and exit
  -t {LIGHT,FLAT,DARK,BIAS}, --data-type {LIGHT,FLAT,DARK,BIAS}
                        data type (default: LIGHT)
  -e EXPTIME, --exptime EXPTIME
                        exposure time in sec
  -f FILTER_NAME, --filter-name FILTER_NAME
                        filter name
  -o OBJECT_NAME, --object-name OBJECT_NAME
                        object name
  -d DIR_DST, --dir-dst DIR_DST
                        destination directory (default: .)

% ./ao2021_s23_01.py -d v0678vir -t LIGHT -e 60.0 -f rp_Astrodon_2019 \
? -o V0678VIR 0214_red/*.fits
Now copying the file "0214_red/lot_20210214_0085_df.fits" to "v0678vir"...
Now copying the file "0214_red/lot_20210214_0086_df.fits" to "v0678vir"...
Now copying the file "0214_red/lot_20210214_0087_df.fits" to "v0678vir"...
Now copying the file "0214_red/lot_20210214_0088_df.fits" to "v0678vir"...
Now copying the file "0214_red/lot_20210214_0089_df.fits" to "v0678vir"...
Now copying the file "0214_red/lot_20210214_0117_df.fits" to "v0678vir"...
```

```
Now copying the file "0214_red/lot_20210214_0118_df.fits" to "v0678vir"...
Now copying the file "0214_red/lot_20210214_0119_df.fits" to "v0678vir"...
Now copying the file "0214_red/lot_20210214_0120_df.fits" to "v0678vir"...
Now copying the file "0214_red/lot_20210214_0121_df.fits" to "v0678vir"...

.....

Now copying the file "0214_red/lot_20210214_0378_df.fits" to "v0678vir"...
Now copying the file "0214_red/lot_20210214_0379_df.fits" to "v0678vir"...
Now copying the file "0214_red/lot_20210214_0380_df.fits" to "v0678vir"...
Now copying the file "0214_red/lot_20210214_0381_df.fits" to "v0678vir"...
Now copying the file "0214_red/lot_20210214_0382_df.fits" to "v0678vir"...
Now copying the file "0214_red/lot_20210214_0386_df.fits" to "v0678vir"...
Now copying the file "0214_red/lot_20210214_0387_df.fits" to "v0678vir"...
Now copying the file "0214_red/lot_20210214_0388_df.fits" to "v0678vir"...
Now copying the file "0214_red/lot_20210214_0389_df.fits" to "v0678vir"...
Now copying the file "0214_red/lot_20210214_0390_df.fits" to "v0678vir"...
% ls v0678vir/
lot_20210214_0085_df.fits               lot_20210214_0241_df.fits
lot_20210214_0086_df.fits               lot_20210214_0257_df.fits
lot_20210214_0087_df.fits               lot_20210214_0258_df.fits
lot_20210214_0088_df.fits               lot_20210214_0259_df.fits
lot_20210214_0089_df.fits               lot_20210214_0260_df.fits
lot_20210214_0117_df.fits               lot_20210214_0261_df.fits
lot_20210214_0118_df.fits               lot_20210214_0277_df.fits
lot_20210214_0119_df.fits               lot_20210214_0278_df.fits
lot_20210214_0120_df.fits               lot_20210214_0279_df.fits
lot_20210214_0121_df.fits               lot_20210214_0280_df.fits

.....

lot_20210214_0198_df.fits               lot_20210214_0379_df.fits
lot_20210214_0217_df.fits               lot_20210214_0380_df.fits
lot_20210214_0218_df.fits               lot_20210214_0381_df.fits
lot_20210214_0219_df.fits               lot_20210214_0382_df.fits
lot_20210214_0220_df.fits               lot_20210214_0386_df.fits
lot_20210214_0221_df.fits               lot_20210214_0387_df.fits
lot_20210214_0237_df.fits               lot_20210214_0388_df.fits
lot_20210214_0238_df.fits               lot_20210214_0389_df.fits
lot_20210214_0239_df.fits               lot_20210214_0390_df.fits
lot_20210214_0240_df.fits
```

3 Making a WCS pipeline

Make a single Python script to set WCS in FITS files. Here is an example.

Python Code 2: ao2021_s23_01.py

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

# importing argparse module
import argparse

# importing datetime module
import datetime

# importing pathlib module
import pathlib
```

```
# importing subprocess module
import subprocess

# importing numpy module
import numpy
import numpy.ma

# importing astropy module
import astropy.convolution
import astropy.coordinates
import astropy.io.fits
import astropy.io.votable
import astropy.units
import astropy.stats

# importing astroquery module
import astroquery.gaia

# importing photutils module
import photutils.segmentation

# importing ssl module
import ssl

# allow insecure downloading
ssl._create_default_https_context = ssl._create_unverified_context

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

# list of astrometric catalogues
list_astrometric_catalogues = ['gaia_edr3', 'gaia_dr2']

# list of sexagesimal format keywords
list_keyword_sexagesimal = ['HA', 'UT', 'ST', 'RA', 'DEC', \
                            'OBJCTRA', 'OBJCTDEC', 'SITELAT', 'SITELONG']

# list of epoch related keywords
list_keyword_epoch = ['EPOCH', 'EQUINOX']

# list of keywords for pixel scale
list_keyword_pixelscale = ['SECPIX1', 'SECPIX2']

# list of keywords for fundamental WCS
list_keyword_wcs = ['CRVAL1', 'CRVAL2', 'CRPIX1', 'CRPIX2', \
                    'CDELT1', 'CDELT2', 'CROTA1', 'CROTA2']

# list of keywords for CD matrix
list_keyword_cdmatrix = ['CD1_1', 'CD1_2', 'CD2_1', 'CD2_2']

# help messages
help_focallength \
    = 'FITS keyword for focal length in mm (default: FOCALLEN)'
help_pixelsize_x \
    = 'FITS keyword for pixel size in X in micron (default: XPIXSZ)'
help_pixelsize_y \
    = 'FITS keyword for pixel size in Y in micron (default: XPIXSZ)'
```

```
help_exptime \
    = 'FITS keyword for integration time in sec (default: EXPTIME)'
help_imwcs \
    = 'location of the command "imwcs" (default: /usr/local/bin/imwcs)'

# constructing parser object
desc = "setting WCS in FITS files"
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-a', '--astrometric-catalogue', \
                     choices=list_astrometric_catalogues, \
                     default='gaia_edr3', \
                     help='choice of astrometric catalogue (default: gaia_edr3')
parser.add_argument ('-e', '--epoch', type=float, default=2000.0, \
                     help='epoch (default: 2000.0)')
parser.add_argument ('-r', '--radius', type=float, default=3.0, \
                     help='search radius for catalogue in FOV (default: 3)')
parser.add_argument ('-s', '--threshold-rejection-sigma', \
                     type=float, default=4.0, \
                     help='sigma-clipping threshold in sigma (default: 4)')
parser.add_argument ('-t', '--threshold-detection-obj', \
                     type=float, default=5.0, \
                     help='detection threshold in sigma (default: 5)')
parser.add_argument ('-u', '--threshold-detection-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 ('-d', '--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 ('-k', '--gaussian-fwhm', type=float, default=3.0, \
                     help='Gaussian FWHM in pixel for convolution (default: 3)')
parser.add_argument ('-b', '--kernel-size', type=int, default=3, \
                     help='Gaussian kernel array size in pixel (default: 3)')
parser.add_argument ('-w', '--command-imwcs', default='/usr/local/bin/imwcs', \
                     help=help_imwcs)
parser.add_argument ('--keyword-exptime', default='EXPTIME', \
                     help=help_exptime)
parser.add_argument ('--keyword-focallength', default='FOCALLEN', \
                     help=help_focallength)
parser.add_argument ('--keyword-pixelsize-x', default='XPIXSZ', \
                     help=help_pixelsize_x)
parser.add_argument ('--keyword-pixelsize-y', default='YPIXSZ', \
                     help=help_pixelsize_y)
parser.add_argument ('FITS_files', nargs='+', help='FITS files')

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

# input parameters
astrometric_catalogue   = args.astrometric_catalogue
epoch                   = args.epoch
radius_fov              = args.radius
threshold_detection_obj = args.threshold_detection_obj
threshold_detection_sky = args.threshold_detection_sky
npixels                 = args.npixels
```

```
dilate_size             = args.dilate_size
maxiters                = args.maxiters
threshold_rejection     = args.threshold_rejection_sigma
gaussian_fwhm           = args.gaussian_fwhm
kernel_array_size       = args.kernel_size
keyword_exptime         = args.keyword_exptime
keyword_focallength     = args.keyword_focallength
keyword_pixelsize_x     = args.keyword_pixelsize_x
keyword_pixelsize_y     = args.keyword_pixelsize_y
command_imwcs           = args.command_imwcs
files_fits              = args.FITS_files

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

# function to fetch a subset of astrometric catalogue
def fetch_astrometric_catalogue (catalogue, ra_deg, dec_deg, radius_deg, \
                                 file_output):
    # command for database query
    if (catalogue == 'gaia_edr3'):
        query_p \
            = "POINT('ICRS',gaiaedr3.gaia_source.ra,gaiaedr3.gaia_source.dec)"
        query_c = "CIRCLE('ICRS',%f,%f,%f)" % (ra_deg, dec_deg, radius_deg)
        query \
            = "SELECT * from gaiaedr3.gaia_source WHERE CONTAINS(%s,%s)=1;" \
            % (query_p, query_c)
    elif (catalogue == 'gaia_dr2'):
        query_p \
            = "POINT('ICRS',gaiadr2.gaia_source.ra,gaiadr2.gaia_source.dec)"
        query_c = "CIRCLE('ICRS',%f,%f,%f)" % (ra_deg, dec_deg, radius_deg)
        query \
            = "SELECT * from gaiadr2.gaia_source WHERE CONTAINS(%s,%s)=1;" \
            % (query_p, query_c)
    # sending a job to Gaia database
    job = astroquery.gaia.Gaia.launch_job_async (query, dump_to_file=True, \
                                                 output_file=file_output, \
                                                 output_format='votable')
    # returning job object
    return (job)

# function to convert vot file to cat file
def convert_from_vot_to_cat (file_vot, file_cat):
    # reading VOTable file
    table_cat = astropy.io.votable.parse_single_table (file_vot).to_table ()
```

```
    # source_id, ra, dec, and r'-band mag
    data_id  = numpy.array (table_cat['source_id'])
    data_ra  = numpy.array (table_cat['ra'])
    data_dec = numpy.array (table_cat['dec'])
    data_r   = numpy.array (table_cat['phot_rp_mean_mag'])

    # writing data of stars in astrometric catalogue into *.cat file
    with open (file_cat, 'w') as fh_cat:
        # writing a header of a local catalogue for WCSTools
        fh_cat.write ("gaia/d/j\n")
        fh_cat.write ("a tiny portion of astrometric catalogue %s\n" \
                      % astrometric_catalogue)
        # writing information of each star to file
        for i in range ( len (data_id) ):
            if ( numpy.isnan (data_r[i]) ):
                # if there is no information of r'-band mag, then use 99.99
                fh_cat.write ("%s %12.8f %+13.8f %7.3f\n" \
                              % (data_id[i], data_ra[i], \
                                 data_dec[i], 99.99 ) )
            else:
                # if there is information of mag, then use mag
                fh_cat.write ("%s %12.8f %+13.8f %7.3f\n" \
                              % (data_id[i], data_ra[i], \
                                 data_dec[i], data_r[i]) )

# function to calculate sky background level
def calc_skybg (image, threshold_detection_sky, npixels, dilate_size, \
                maxiters, threshold_rejection):
    # masking stars
    source_mask \
        = photutils.segmentation.make_source_mask (image, \
                                                   threshold_detection_sky, \
                                                   npixels=npixels, \
                                                   sigclip_iters=maxiters, \
                                                   dilate_size=dilate_size)
    # making a masked array
    image_masked = numpy.ma.array (image, mask=source_mask)
    # sky background estimate using sigma-clipping
    skybg_mean, skybg_median, skybg_stddev \
        = astropy.stats.sigma_clipped_stats (image_masked, \
                                             sigma=threshold_rejection)
    # mode calculation using empirical formula
    skybg_mode = 3.0 * skybg_median - 2.0 * skybg_mean
    # returning estimated sky background level and its standard deviation
    return (skybg_mode, skybg_stddev)

def do_source_extraction (image, skybg_mode, skybg_stddev, \
                          threshold_detection_obj, \
                          gaussian_fwhm, kernel_array_size):
    # detection threshold in ADU
    threshold_detection_adu \
        = skybg_mode + threshold_detection_obj * 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_detection_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_sources \
        = photutils.segmentation.SourceCatalog (image, image_deblend)
    # making a table
    table_sources = catalogue_sources.to_table ()
    # returning source table
    return (table_sources)

#
# main routine
#

# check of command "imwcs"
path_imwcs = pathlib.Path (command_imwcs)
if not ( path_imwcs.exists () and path_imwcs.is_file () ):
    print ("Cannot find the command \"imwcs\"!")
    sys.exit ()

# processing each FITS file
for file_fits in files_fits:
    # making pathlib object
    path_fits = pathlib.Path (file_fits)

    # file names
    file_wcs = path_fits.stem + 'w.fits'
    file_log = path_fits.stem + 'w.log'
    file_vot = path_fits.stem + 'w.vot.gz'
    file_cat = path_fits.stem + 'w.cat'
    file_xym = path_fits.stem + 'w.xym'

    print ('FITS file =', file_fits)
    print ('wcs file  =', file_wcs)
    print ('log file  =', file_log)
    print ('vot file  =', file_vot)
    print ('cat file  =', file_cat)
    print ('xym file  =', file_xym)

    # check of FITS file name
    if not ( path_fits.exists () and path_fits.is_file () \
             and (file_fits[-5:] == '.fits') ):
        # writing an error message to log file
        with open (file_log, 'w') as fh_log:
            fh_log.write ("Input file \"%s\" is missing, " % file_fits)
            fh_log.write ("or is not a regular file,\n")
            fh_log.write ("or is not a FITS file (*.fits)\n")
            fh_log.write ("Check the file.\n")
        # skip processing the file
        continue
```

```
    # reading FITS file
    (header, image) = read_fits (file_fits)

    # important parameters of FITS file
    exptime = header[keyword_exptime]

    # reformatting keyword values
    #   e.g. '12 34 56.7' ==> '12:34:56.7'
    for keyword in list_keyword_sexagesimal:
        # if the keyword exists, then process
        if (keyword in header):
            # value
            value = header[keyword]
            # if value is not '12:34:56.7' format, then reformat the string
            if not (':' in value):
                # modifying the keyword value
                header[keyword] = value.replace (' ', ':')

    # calculation of pixel scale
    pixelsize_x_micron  = header[keyword_pixelsize_x]
    pixelsize_y_micron  = header[keyword_pixelsize_y]
    focallength_mm      = header[keyword_focallength]
    pixelsize_x_m       = pixelsize_x_micron * 10**-6
    pixelsize_y_m       = pixelsize_y_micron * 10**-6
    focallength_m       = focallength_mm * 10**-3
    pixelscale_x_deg    = pixelsize_x_m / focallength_m * 180.0 / numpy.pi
    pixelscale_y_deg    = pixelsize_y_m / focallength_m * 180.0 / numpy.pi
    pixelscale_x_arcsec = pixelscale_x_deg * 3600.0
    pixelscale_y_arcsec = pixelscale_y_deg * 3600.0

    # adding SECPIX1 and SECPIX2 keywords to FITS header
    for keyword in list_keyword_pixelscale:
        # SECPIX1
        if (keyword[-1] == '1'):
            header[keyword] = pixelscale_x_arcsec * -1.0
        # SECPIX2
        elif (keyword[-1] == '2'):
            header[keyword] = pixelscale_y_arcsec

    # adding EPOCH and EQUINOX keywords to FITS header
    for keyword in list_keyword_epoch:
        # header['EPOCH'] = header['EQUINOX'] = epoch (e.g. 2000.0)
        header[keyword] = epoch

    # number of pixels in X and Y
    if ('NAXIS1' in header):
        npix_x = header['NAXIS1']
    else:
        # writing an error message to log file
        with open (file_log, 'w') as fh_log:
            fh_log.write ("NAXIS1 keyword is missing in %s." % file_fits)
            fh_log.write ("Check the file.\n")
        # skip processing the file
        continue
    if ('NAXIS2' in header):
        npix_y = header['NAXIS2']
    else:
        # writing an error message to log file
```

```
        with open (file_log, 'w') as fh_log:
            fh_log.write ("NAXIS2 keyword is missing in %s." % file_fits)
            fh_log.write ("Check the file.\n")
        # skip processing the file
        continue

    # field-of-view
    fov_x_arcsec = pixelscale_x_arcsec * npix_x
    fov_y_arcsec = pixelscale_y_arcsec * npix_y
    if (fov_x_arcsec > fov_y_arcsec):
        fov_arcsec = fov_x_arcsec
    else:
        fov_arcsec = fov_y_arcsec
    fov_deg = fov_arcsec / 3600.0

    # RA and Dec
    ra_str  = header['RA']
    dec_str = header['DEC']

    # making SkyCoord object
    coord = astropy.coordinates.SkyCoord (ra_str, dec_str, \
                                          unit=(u_ha, u_deg), frame='icrs')
    coord_ra_deg  = coord.ra.deg
    coord_dec_deg = coord.dec.deg

    # adding initial guess of WCS related keywords
    # a flag for WCS
    have_wcs = 'YES'
    # check of CD matrix keywords
    for keyword in list_keyword_cdmatrix:
        # if there is no CD matrix keyword, then set have_wcs = 'NO'
        if not (keyword in header):
            have_wcs = 'NO'
    # if there is no WCS in FITS header, then add initial guess values
    if (have_wcs == 'NO'):
        # adding WCS related keywords in FITS header
        for keyword in list_keyword_wcs:
            if (keyword == 'CRVAL1'):
                header[keyword] = coord_ra_deg
            elif (keyword == 'CRVAL2'):
                header[keyword] = coord_dec_deg
            elif (keyword == 'CRPIX1'):
                header[keyword] = npix_x / 2.0
            elif (keyword == 'CRPIX2'):
                header[keyword] = npix_y / 2.0
            elif (keyword == 'CDELT1'):
                header[keyword] = pixelscale_x_deg * -1.0
            elif (keyword == 'CDELT2'):
                header[keyword] = pixelscale_y_deg * -1.0
            elif (keyword == 'CROTA1'):
                header[keyword] = 0.0
            elif (keyword == 'CROTA2'):
                header[keyword] = 0.0

    # writing a FITS file
    now = datetime.datetime.now ()
    header['comment'] = " "
    header['comment'] = "Update on %s." % now
    header['comment'] = "Some keywords needed for WCS are added."
```

```
    header['comment'] = "New file name = %s" % file_wcs
    astropy.io.fits.writeto (file_wcs, image, header=header)

    # retrieving a subset of astrometric catalogue
    radius_deg = fov_deg * radius_fov
    job = fetch_astrometric_catalogue (astrometric_catalogue, \
                                       coord_ra_deg, coord_dec_deg, \
                                       radius_deg, file_vot)

    # reading VOTable file and write *.cat file for WCSTools
    convert_from_vot_to_cat (file_vot, file_cat)

    # sky background level estimate
    skybg_mode, skybg_stddev = calc_skybg (image, threshold_detection_sky, \
                                           npixels, dilate_size, maxiters, \
                                           threshold_rejection)

    # source extraction
    table_sources = do_source_extraction (image, skybg_mode, skybg_stddev, \
                                          threshold_detection_obj, \
                                          gaussian_fwhm, kernel_array_size)

    # writing daofind format file of detected sources
    with open (file_xym, 'w') as fh_xym:
        # for each object
        for i in range ( len (table_sources) ):
            # X and Y coordinate
            x_c = table_sources['xcentroid'][i]
            y_c = table_sources['ycentroid'][i]
            # calculation of instrumental magnitude
            net_flux = table_sources['kron_flux'][i]
            inst_mag = -2.5 * numpy.log10 (net_flux / exptime)
            # writing X, Y, and instrumental magnitude
            fh_xym.write ("%10.3f %10.3f %+7.3f\n" % (x_c, y_c, inst_mag) )


    # setting WCS of a FITS file using WCSTools
    # number of execution of imwcs command
    n_exe = 5
    # initial tolerance of fitting in pixel
    tolerance_ini = 9
    # iterations
    for i in range (n_exe):
        # tolerance of fitting in pixel
        tolerance = tolerance_ini - 2 * i
        # command
        command_wcs = "%s -v -o -h 200 -t %d -d %s -c %s %s" \
            % (command_imwcs, tolerance, file_xym, file_cat, file_wcs)
        # executing command
        subprocess.run (command_wcs, shell=True)
```

Run the script, and carry out astrometric calibration and set WCS in FITS files.

```
% chmod a+x ao2021_s23_02.py
% ./ao2021_s23_02.py -h
Created TAP+ (v20200428.1) - Connection:
        Host: gea.esac.esa.int
        Use HTTPS: True
```

```
        Port: 443
        SSL Port: 443
Created TAP+ (v20200428.1) - Connection:
        Host: gea.esac.esa.int
        Use HTTPS: True
        Port: 443
        SSL Port: 443
usage: ao2021_s23_02.py [-h] [-a {gaia_edr3,gaia_dr2}] [-e EPOCH] [-r RADIUS]
                        [-s THRESHOLD_REJECTION_SIGMA]
                        [-t THRESHOLD_DETECTION_OBJ]
                        [-u THRESHOLD_DETECTION_SKY] [-n NPIXELS]
                        [-d DILATE_SIZE] [-m MAXITERS] [-k GAUSSIAN_FWHM]
                        [-b KERNEL_SIZE] [-w COMMAND_IMWCS]
                        [--keyword-exptime KEYWORD_EXPTIME]
                        [--keyword-focallength KEYWORD_FOCALLENGTH]
                        [--keyword-pixelsize-x KEYWORD_PIXELSIZE_X]
                        [--keyword-pixelsize-y KEYWORD_PIXELSIZE_Y]
                        FITS_files [FITS_files ...]

setting WCS in FITS files

positional arguments:
  FITS_files            FITS files

optional arguments:
  -h, --help            show this help message and exit
  -a {gaia_edr3,gaia_dr2}, --astrometric-catalogue {gaia_edr3,gaia_dr2}
                        choice of astrometric catalogue (default: gaia_edr3
  -e EPOCH, --epoch EPOCH
                        epoch (default: 2000.0)
  -r RADIUS, --radius RADIUS
                        search radius for catalogue in FOV (default: 3)
  -s THRESHOLD_REJECTION_SIGMA, --threshold-rejection-sigma THRESHOLD_REJECTION_
SIGMA
                        sigma-clipping threshold in sigma (default: 4)
  -t THRESHOLD_DETECTION_OBJ, --threshold-detection-obj THRESHOLD_DETECTION_OBJ
                        detection threshold in sigma (default: 5)
  -u THRESHOLD_DETECTION_SKY, --threshold-detection-sky THRESHOLD_DETECTION_SKY
                        detection threshold for sky estimate (default: 2)
  -n NPIXELS, --npixels NPIXELS
                        minimum number of pixels for detection (default: 5)
  -d DILATE_SIZE, --dilate-size DILATE_SIZE
                        dilate size (default: 21)
  -m MAXITERS, --maxiters MAXITERS
                        maximum number of iterations (default: 30)
  -k GAUSSIAN_FWHM, --gaussian-fwhm GAUSSIAN_FWHM
                        Gaussian FWHM in pixel for convolution (default: 3)
  -b KERNEL_SIZE, --kernel-size KERNEL_SIZE
                        Gaussian kernel array size in pixel (default: 3)
  -w COMMAND_IMWCS, --command-imwcs COMMAND_IMWCS
                        location of the command "imwcs" (default:
                        /usr/local/bin/imwcs)
  --keyword-exptime KEYWORD_EXPTIME
                        FITS keyword for integration time in sec (default:
                        EXPTIME)
  --keyword-focallength KEYWORD_FOCALLENGTH
                        FITS keyword for focal length in mm (default:
                        FOCALLEN)
  --keyword-pixelsize-x KEYWORD_PIXELSIZE_X
```

```
                        FITS keyword for pixel size in X in micron (default:
                        XPIXSZ)
  --keyword-pixelsize-y KEYWORD_PIXELSIZE_Y
                        FITS keyword for pixel size in Y in micron (default:
                        XPIXSZ)

% ls
0214_red@           ao2021_s23_01.py~*  ao2021_s23_02.py~*
ao2021_s23_01.py*   ao2021_s23_02.py*   v0678vir/
% ./ao2021_s23_02.py v0678vir/lot_20210214_*_df.fits
% ls -l *_dfw.fits
-rw-r--r--  1 daisuke  taiwan  33566400 Jun 10 20:26 lot_20210214_0085_dfw.fits
-rw-r--r--  1 daisuke  taiwan  33566400 Jun 10 20:27 lot_20210214_0086_dfw.fits
-rw-r--r--  1 daisuke  taiwan  33566400 Jun 10 20:27 lot_20210214_0087_dfw.fits
-rw-r--r--  1 daisuke  taiwan  33566400 Jun 10 20:28 lot_20210214_0088_dfw.fits
-rw-r--r--  1 daisuke  taiwan  33566400 Jun 10 20:29 lot_20210214_0089_dfw.fits
-rw-r--r--  1 daisuke  taiwan  33569280 Jun 10 20:29 lot_20210214_0117_dfw.fits
-rw-r--r--  1 daisuke  taiwan  33569280 Jun 10 20:30 lot_20210214_0118_dfw.fits
-rw-r--r--  1 daisuke  taiwan  33569280 Jun 10 20:30 lot_20210214_0119_dfw.fits
-rw-r--r--  1 daisuke  taiwan  33569280 Jun 10 20:31 lot_20210214_0120_dfw.fits
-rw-r--r--  1 daisuke  taiwan  33566400 Jun 10 20:31 lot_20210214_0121_dfw.fits

.....

-rw-r--r--  1 daisuke  taiwan  33569280 Jun 10 21:02 lot_20210214_0378_dfw.fits
-rw-r--r--  1 daisuke  taiwan  33566400 Jun 10 21:03 lot_20210214_0379_dfw.fits
-rw-r--r--  1 daisuke  taiwan  33566400 Jun 10 21:03 lot_20210214_0380_dfw.fits
-rw-r--r--  1 daisuke  taiwan  33566400 Jun 10 21:04 lot_20210214_0381_dfw.fits
-rw-r--r--  1 daisuke  taiwan  33566400 Jun 10 21:04 lot_20210214_0382_dfw.fits
-rw-r--r--  1 daisuke  taiwan  33566400 Jun 10 21:04 lot_20210214_0386_dfw.fits
-rw-r--r--  1 daisuke  taiwan  33566400 Jun 10 21:05 lot_20210214_0387_dfw.fits
-rw-r--r--  1 daisuke  taiwan  33566400 Jun 10 21:05 lot_20210214_0388_dfw.fits
-rw-r--r--  1 daisuke  taiwan  33566400 Jun 10 21:06 lot_20210214_0389_dfw.fits
-rw-r--r--  1 daisuke  taiwan  33566400 Jun 10 21:06 lot_20210214_0390_dfw.fits
```

4 RA and Dec of V0678 Vir

Download GCVS version 5.1.

```
% curl -o gcvs5.data http://www.sai.msu.su/gcvs/gcvs/gcvs5/gcvs5.txt
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 12.9M  100 12.9M    0     0   372k      0  0:00:35  0:00:35 --:--:--  585k
% ls -l gcvs5.data
-rw-r--r--  1 daisuke  taiwan  13608033 Jun 10 21:22 gcvs5.data
```

Search for the star V0678 Vir.

```
% grep 'V0678 Vir' gcvs5.data | cut -b -80
860678 |V0678 Vir  |123148.11 -020602.3 |EW        | 14.74   |  15.13     |  15.
```

RA and Dec of V0678 Vir on GCVS 5.1 are 12:31:48.11 and -02:06:02.3, respectively.

5 Position of V0678 Vir on Gaia EDR3

Check the position of V0678 Vir on Gaia EDR3.

```
% scat -r 3 lot_20210214_0085_dfw.cat 12:31:48.11 -02:06:02.3
-000000002147483648 12:31:48.103 -02:06:02.24    14.21    0.12
```

RA and Dec of V0678 Vir on Gaia EDR3 are 12:31:48.103 and -02:06:02.24, respectively.

6 Measuring position of V0678 Vir on images

Make a Python script to measure the position of V0678 Vir on images.

Python Code 3: ao2021_s23_03.py

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

# importing argparse module
import argparse

# importing sys module
import sys

# importing pathlib module
import pathlib

# importing numpy module
import numpy

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

# choice of PSF model
list_psf_model = ['2dg', '2dm']

# constructing parser object
desc = "measuring position on images"
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-m', '--psf-model', choices=list_psf_model, \
                     default='2dg', \
                     help='PSF model [2dg=Gaussian, 2dm=Moffat] (default: 2dg)')
parser.add_argument ('-p', '--position', default='', \
                     help='RA and Dec in hh:mm:ss.ss,+/-dd:mm:ss.s format')
parser.add_argument ('-w', '--half-width', type=int, default=5, \
                     help='half-width of centroid box (default: 30)')
parser.add_argument ('FITS_files', nargs='+', help='FITS files')

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

# input parameters
psf_model  = args.psf_model
position   = args.position
```

```
half_width = args.half_width
files_fits = args.FITS_files

# check of input position
if not ( (',' in position) and (':' in position) ):
    print ("RA and Dec must be given in hh:mm:ss.ss,+/-dd:mm:ss.s format.", \
           file=sys.stderr)
    sys.exit ()

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

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

# RA and Dec
ra_str, dec_str = position.split (',')

# making SkyCoord object
coord = astropy.coordinates.SkyCoord (ra_str, dec_str, \
                                      unit=(u_ha, u_deg), frame='icrs')

# RA and Dec in deg
ra_deg  = coord.ra.deg
dec_deg = coord.dec.deg

# processing each FITS file
for file_fits in files_fits:
    # making pathlib object
    path_fits = pathlib.Path (file_fits)
    # check of FITS file
    if not ( (path_fits.exists () ) and (path_fits.is_file () ) \
             and (file_fits[-5:] == '.fits') ):
        print ("Input file must be a FITS file.", file=sys.stderr)
        continue
    # reading FITS file
    (header, image) = read_fits (file_fits)
    # WCS
    wcs = astropy.wcs.WCS (header)
    # calculation of (x, y) from RA and Dec
    (x, y) = wcs.world_to_pixel (coord)

    # region of calculation
    box_xmin = int (x) - half_width
```

```
    box_xmax = int (x) + half_width
    box_ymin = int (y) - half_width
    box_ymax = int (y) + half_width
    box = image[box_ymin:box_ymax, box_xmin:box_xmax]

    # rough background subtraction
    box -= numpy.median (box)

    # PSF fitting
    box_y, box_x = numpy.indices (box.shape)
    if (psf_model == '2dg'):
        psf_init = astropy.modeling.models.Gaussian2D (x_mean=half_width, \
                                                       y_mean=half_width)
    elif (psf_model == '2dm'):
        psf_init = astropy.modeling.models.Moffat2D (x_0=half_width, \
                                                     y_0=half_width)
    fit = astropy.modeling.fitting.LevMarLSQFitter ()
    psf_fitted = fit (psf_init, box_x, box_y, box, maxiter=1000)

    # results of fitting
    if (psf_model == '2dg'):
        x_centre = psf_fitted.x_mean.value
        y_centre = psf_fitted.y_mean.value
        fwhm_x   = psf_fitted.x_fwhm
        fwhm_y   = psf_fitted.y_fwhm
        fwhm     = (fwhm_x + fwhm_y) / 2.0
    elif (psf_model == '2dm'):
        x_centre = psf_fitted.x_0.value
        y_centre = psf_fitted.y_0.value
        fwhm     = psf_fitted.fwhm

    # (x, y) of centre of star
    x_c = x_centre + box_xmin
    y_c = y_centre + box_ymin

    # measured RA and Dec
    coord_measured = wcs.pixel_to_world (x_c, y_c)

    # printing result
    print ("%-26s %12.8f %+13.8f %-32s # (x,y)=(%10.4f,%10.4f)" \
           % (path_fits.name, coord_measured.ra.deg, coord_measured.dec.deg, \
              coord_measured.to_string ('hmsdms'), x_c, y_c) )
```

Execute the script, and measure the position of V0678 Vir.

```
% chmod a+x ao2021_s23_03.py
% ./ao2021_s23_03.py -h
usage: ao2021_s23_03.py [-h] [-m {2dg,2dm}] [-p POSITION] [-w HALF_WIDTH]
                        FITS_files [FITS_files ...]

measuring position on images

positional arguments:
  FITS_files            FITS files

optional arguments:
  -h, --help            show this help message and exit
  -m {2dg,2dm}, --psf-model {2dg,2dm}
```

```
                        PSF model [2dg=Gaussian, 2dm=Moffat] (default: 2dg)
  -p POSITION, --position POSITION
                        RA and Dec in hh:mm:ss.ss,+/-dd:mm:ss.s format
  -w HALF_WIDTH, --half-width HALF_WIDTH
                        half-width of centroid box (default: 30)

% ./ao2021_s23_03.py -p 12:31:48.103,-02:06:02.24 *_dfw.fits \
? > v0678vir_astrometry.data
% head -10 v0678vir_astrometry.data | cut -b -80
lot_20210214_0085_dfw.fits 187.95028975   -2.10069185 12h31m48.0695s -02d06m02.4
lot_20210214_0086_dfw.fits 187.95028946   -2.10069371 12h31m48.0695s -02d06m02.4
lot_20210214_0087_dfw.fits 187.95029333   -2.10071070 12h31m48.0704s -02d06m02.5
lot_20210214_0088_dfw.fits 187.95030217   -2.10069590 12h31m48.0725s -02d06m02.5
lot_20210214_0089_dfw.fits 187.95029520   -2.10069686 12h31m48.0708s -02d06m02.5
lot_20210214_0117_dfw.fits 187.95033475   -2.10069649 12h31m48.0803s -02d06m02.5
lot_20210214_0118_dfw.fits 187.95033349   -2.10069595 12h31m48.08s -02d06m02.505
lot_20210214_0119_dfw.fits 187.95033817   -2.10069817 12h31m48.0812s -02d06m02.5
lot_20210214_0120_dfw.fits 187.95033499   -2.10069434 12h31m48.0804s -02d06m02.4
lot_20210214_0121_dfw.fits 187.95034780   -2.10069607 12h31m48.0835s -02d06m02.5
```

7 Calculating mean value and standard deviation

Make a Python script to calculate mean value and standard deviation of position of V0678 Vir.

Python Code 4: ao2021_s23_04.py

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

# importing numpy module
import numpy
import numpy.ma

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

# data file
file_data = 'v0678vir_astrometry.data'

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

# making empty Numpy arrays
array_ra = numpy.array ([])
array_dec = numpy.array ([])

# opening data file
with open (file_data, 'r') as fh:
    # reading data file
    for line in fh:
        # splitting data
        records = line.split ()
        # RA in deg
        ra_deg = float (records[1])
        # Dec in deg
        dec_deg = float (records[2])
```

```
        # appending data to Numpy arrays
        array_ra  = numpy.append (array_ra, ra_deg)
        array_dec = numpy.append (array_dec, dec_deg)

# sigma-clipping
array_ra_masked  = astropy.stats.sigma_clip (array_ra, sigma=3.0, \
                                             maxiters=10, cenfunc='median', \
                                             masked=True)
array_dec_masked = astropy.stats.sigma_clip (array_dec, sigma=3.0, \
                                             maxiters=10, cenfunc='median', \
                                             masked=True)

# printing results of sigma-clipping
print (array_ra_masked)
print (array_dec_masked)

# mean and standard deviation
ra_mean    = numpy.ma.mean (array_ra_masked)
ra_stddev  = numpy.ma.std (array_ra_masked)
dec_mean   = numpy.ma.mean (array_dec_masked)
dec_stddev = numpy.ma.std (array_dec_masked)

# coordinate
coord = astropy.coordinates.SkyCoord (ra_mean, dec_mean, \
                                      unit=(u_deg, u_deg), frame='icrs')
coord_str = coord.to_string ('hmsdms')
(coord_ra_str, coord_dec_str) = coord_str.split ()

# printing results
print ("RA  = %12.8f deg = %s" % (ra_mean, coord_ra_str) )
print ("Dec = %12.8f deg = %s" % (dec_mean, coord_dec_str) )
print ("stddev of RA  = %6.3f arcsec" % (ra_stddev * 3600.0) )
print ("stddev of Dec = %6.3f arcsec" % (dec_stddev * 3600.0) )
```

Run the script, and calculate mean and standard deviation.

```
% chmod a+x ao2021_s23_04.py
% ./ao2021_s23_04.py
[-- -- -- -- -- 187.95033475 187.95033349 187.95033817 187.95033499
 187.9503478 187.95037699 187.95038757 187.95037962 187.95037973
 187.95037467 187.95034133 187.95033852 187.95035199 187.95033621
 187.9503455 187.95035916 187.95035875 187.95035361 187.95035441
 187.9503492 187.95036451 187.95035949 187.95035312 187.95035865
 187.95035237 187.95035424 187.95035434 187.95035374 187.95036089 --
 187.95035686 187.95035071 187.95035213 187.95035393 187.9503535
 187.95034871 187.95035393 187.95035512 187.95035324 187.95035352
 187.95035682 187.95035432 187.95036195 187.9503549 187.95036007
 187.95035701 187.95035748 187.95035618 -- 187.95035634 187.95035566
 187.95036 187.95035678 187.95036622 187.95035792 187.95035729
 187.95035975 187.95036519 187.95036729 187.95036464 187.95037276
 187.95037021 187.95037276 187.95036333 -- 187.95036077 187.95036573
 187.95036927 187.95036834 187.95036585 187.95036606 187.95037363]
[-2.10069185 -2.10069371 -2.1007107 -2.1006959 -2.10069686 -2.10069649
 -2.10069595 -2.10069817 -2.10069434 -2.10069607 -2.1006931 -2.10069079
 -2.10070022 -2.10069484 -2.10070021 -2.10071007 -2.10071235 -2.10070954
 -2.10072073 -2.10071706 -2.10071149 -2.10071436 -2.1007132 -2.10070545
 -2.10070535 -2.10070208 -2.10070712 -2.10070009 -2.10069452 -2.10069723
 -2.10070614 -2.10070404 -2.10070574 -2.10071106 -- -2.10071002
```

```
 -2.10070404 -2.10070001 -2.10069747 -2.10069643 -2.10069584 -2.10069357
 -2.10069351 -2.10069567 -2.10069593 -2.10068856 -2.10069543 -2.10069794
 -2.10069638 -2.10068958 -2.10069744 -2.10069611 -2.10069369 --
 -2.10069842 -2.10069579 -2.10069337 -2.10069726 -2.10068853 -2.10069278
 -2.10069965 -2.10069951 -2.10072755 -2.10072359 -2.10072774 -2.10072835
 -2.10072918 -2.10070741 -2.10071776 -- -2.10071243 -2.10070527
 -2.10071215 -2.1007139 -2.10070801 -2.1007099 -2.10070461]
RA  = 187.95035788 deg = 12h31m48.0859s
Dec =  -2.10070308 deg = -02d06m02.5311s
stddev of RA  =  0.039 arcsec
stddev of Dec =  0.036 arcsec
```

8 For your training

Read followings.

1. WCSTools
   - http://tdc-www.harvard.edu/software/wcstools/
   - http://tdc-www.harvard.edu/software/wcstools/publications/adass2018abs.html
   - http://tdc-www.harvard.edu/software/wcstools/publications/wcstools.adass98.html
2. FITS World Coordinate System (WCS)
   - https://fits.gsfc.nasa.gov/fits_wcs.html
3. FITS World Coordinate Systems
   - https://www.atnf.csiro.au/people/mcalabre/WCS/
4. astropy.coordinates
   - https://docs.astropy.org/en/stable/coordinates/
5. astropy.io.fits
   - https://docs.astropy.org/en/stable/io/fits/
6. astropy.io.votable
   - https://docs.astropy.org/en/stable/io/votable/
7. astropy.modeling
   - https://docs.astropy.org/en/stable/modeling/
8. astropy.stats
   - https://docs.astropy.org/en/stable/stats/
9. astropy.units
   - https://docs.astropy.org/en/stable/units/
10. astropy.wcs
    - https://docs.astropy.org/en/stable/wcs/
11. numpy.ma
    - https://numpy.org/doc/stable/reference/maskedarray.html
    - https://numpy.org/doc/stable/reference/maskedarray.generic.html
12. photutils.segmentation

- `https://photutils.readthedocs.io/en/stable/segmentation.html`

13. argparse
    - `https://docs.python.org/3/library/argparse.html`
14. datetime
    - `https://docs.python.org/3/library/datetime.html`
15. pathlib
    - `https://docs.python.org/3/library/pathlib.html`
16. shutil
    - `https://docs.python.org/3/library/shutil.html`
17. subprocess
    - `https://docs.python.org/3/library/subprocess.html`

9 Assignment

None.