ommosaic

June 2, 2019

Abstract

Ommosaic stacks a list of **OM product** images (sky or unrotated, high or low resolution). If required, the program will attempt to align the images before stacking. Sky-images from different observations of the same target object can be stacked. Source-detection, using **omdetect**, can be performed on the output image.

1 Instruments/Modes

Instrument	Mode
OM	IMAGING

2 Use

pipeline processing	yes
interactive analysis	yes

3 Description

ommosaic accepts an arbitrary number of sky-image files (ie those produced by **omatt**), or product unrotated-images (ie those produced by **ommodmap**) and produces a mosaiced image. **The latter is not recommended for precision data analysis.** The task can process a mixture of both low and high-resolution images. The output image file will contain three images:

1. The stacked image stored in the **PRIMARY** block of the **FITS** file.
2. An exposure image stored in the **EXPOSURE** block of the **FITS** file. It is constructed using the individual exposure times from each input file. The exposure times are corrected for the **OM** deadtime-fraction time before the image is created.
3. A 16-bit quality image stored in the **QUALITY** block of the **FITS** file.

The output image file (parameter **mosaicedset**) can be processed by **omdetect**. It should be noted that in **omdetect's** photometry stage, no coincidence-loss corrections will be applied (since the frame-time and deadtime-fraction parameters of the mosaiced images vary from one exposure to the other) and this means that the photometric accuracy will decrease with increasing source-brightness.

3.1 Sky-images- Astrometry

Omatt produces the sky-images and computes the celestial coordinates of the sources. Additionally, in order to correct for any small error in the **OM** pointing direction, leading to small **RA** and **Dec** offsets, it can also try to correct the source coordinates for these offsets by using the USNO-B catalogue. If this astrometry correction is successful, **omatt** will set a FITS key-word **POSCOROK** to **TRUE** in the headers of the source-list and sky-image files and **RAOFFSET** and **DEOFFSET** to the computed **RA** and **Dec** offsets. The values of **CRVAL1** and **CRVAL2** will also be corrected. If it fails, **POSCOROK** will be set to **FALSE** and the other key-words to zero.

For each input sky-image **ommosaic** will check the **POSCOROK** keyword and if it equal to **FALSE**, if possible it will correct the reference pixel coordinates (**CRVAL1** and **CRVAL2**) by interpolation or extrapolation using the successful correction values. It will add the FITS keyword **POSINTRP** to the sky-image header, and it will be set to **TRUE** if an interpolation/extrapolation has been done, or **FALSE** otherwise. The keywords **RA_OFF** and **DEC_OFF** will also be added giving the interpolated/extrapolated values (0.0 if no interpolation or extrapolation).

When **ommosaic** is run with its parameter **mincorr** set to 0.0 (default value- no image-alignment attempted) the astrometric accuracy of the computed stacked image will be determined by the accuracies of the individual reference-pixel coordinates (**CRVAL1** and **CRVAL2**), which in turn will depend on the accuracies of the computed astrometric correction values. A single bad computed **RA** or **Dec** offset can have an adverse effect on the stacked image and astrometry. Therefore, in order to try to warn the user of any possible problems, **ommosaic** will analyse the computed **RA** and **Dec** offsets (using a **MAD** test described later on) and warn the user of any possible discrepant values.

If none of the input sky images had a successful astrometric correction (**POSCOROK=false**), then the keyword **POSCOROK** in the **FITS** header of the output image will be set to **FALSE**. In this case the **astrometry** contains the uncertainty due to the unknown error in the **OM** pointing direction.

If at least one of input sky images had a successful astrometric correction (**POSCOROK=true**), then the keyword **POSCOROK** in the **FITS** header of the output image will be set to **TRUE** and the keywords **RAOFFSET** and **DEOFFSET** to the mean **RAOFFSET** and **DEOFFSET** values from all the input images, respectively (some of which may have been computed by interpolation or extrapolation.)

However, a successful astrometric correction may have been done by omsrclistcomb.- this is because there may be exposure source-lists containing too few sources for an astrometric correction to be performed by **omatt** (particularly in the UV) , but in the observation source-list produced by **omsrclistcomb** there may be sufficient sources for a successful correction.

4 Parameters

This section documents the parameters recognized by this task (if any).

Parameter	Mand	Type	Default	Constraints
imagesets	yes	string	none	

List of input image datasets to be mosaiced

mosaicedset	yes	string	none	

Name of output mosaiced image dataset

correlset	no	string	none	

Name of output correlation file dataset

binaxis	no	real	0	0 : 2

Image binning factor. 0 means take the largest binning factor of all the input images and compute the output image using this value. 1 means that the output image will be a high-resolution image, 2 a low-resoloution one.

mincorr	no	real	0	-1 to +1

Minimum acceptable correlation successful image alignment If 0 then no alignment attempted. If ¡ 0 then images stacked even if the correlation is less than abs(mincorr)

The following parameters are for use in the image-alignment algorithm- the default values have been set by trial and error- changing any of them may significantly increase the cpu time without improving anything.

nsigma	no	real	2	0.5 :

Image pixel used in cross-correlation if the pixel value is at least background + max(1.0, nsigma * sqrt(background)), where background is the computed background level at the pixel

minfraction	no	real	0.5	0.05 : 1

Minimum acceptable area overlap of 2 images before alignment is attempted

maxdx	no	int	5	1 : 50

Maximum x and y offsets (in image pixels) between 2 images refereence pixels to be used in cross-correlation

numintervals	no	int	2	1 : 100

The number of intervals to split an x or y offset value into. Ie the total number of x or y offsets to be used to compute the initial correlation coefficients will be (2xdx+1)*numintervals and (2xdy+1)*numintervals,respectively.

di	no	int	10	5 to 100

The correlation image will be 2*di+1 pixels square.

minnumpixels	no	int	100	20 :

The minimum number of image pixels that lie above the background than can be used in the cross-correlation.

5 Usage

Ommosaic can work in one of three different ways, which are described in the following sections;

5.1 Stacking of images without any internal alignment

The following command-line, **ommosaic imagesets="image1 image2 ..." mosaicedset=mosaicedimage.fits mincorr=0.0**, will stack the images image1, image2,**without** without trying to align them, and store the stacked image in the file **mosaicedimage.fits**

5.2 Stacking of images, but only the ones that have been successfully aligned internally

The method used to align two images is based on normalized cross-correlation, and is a widely used method in many disciplines, including astronomy (please see the article "Proceedings of the SPIE, Advanced Signal Processing Algorithms, Architectures, and Implementations XV1, Vol. 6313, pp. D1-D13, San Diego, CA, Aug. 2006.) **It should be noted that this functionality is new in Version 2, and since only a limited amount of testing has been done it should be used with caution. The correlation images should be examined to check that there is a good peak and the computed offsets checked.**

The following command-line, **ommosaic imagesets="image1 image2 ..." mosaicedset=mosaicedimage.fits mincorr=0.5 correlset=correl.fits** attempts to align the images image1, image2, stacks the images that have been successfully aligned (ie those with a computed correlation coefficent of at least equal to 0.5, stores the stacked image in the file **mosaicedimage.fits** and the correlation images in the file **correl.fits**.

5.3 Stacking of images with internal image alignment

The following command-line, **ommosaic imagesets="image1 image2 ..." mosaicedset=mosaicedimage.fits mincorr=-0.5 correlset=correl.fits** stacks the images image1, image2, and stores the stacked image

in the file **mosaicedimage.fits** and the correlation images in the file **correl.fits**.

The difference between this command-line and the previous one is that the **negative mincorr** value indicates that all the images will be stacked, regardless of the computed correlation coefficient.

6 Errors

This section documents warnings and errors generated by this task (if any). Note that warnings and errors can also be generated in the SAS infrastructure libraries, in which case they would not be documented here. Refer to the index of all errors and warnings available in the HTML version of the SAS documentation.

You are combining identical images! *(warning)*

(warning)

corrective action:
corrective action: file nameYou are combining images from different filters!List of filters found

You are combining images from the same exposure! *(warning)*

(warning)

corrective action:
corrective action: List of overlapping files foundYou are combining images from the same exposure!Checking the list of input files

Non of the images are astrometry-corrected! *(warning)*

corrective action: None

7 Input Files

1. List of image files (image in primary header)

8 Output Files

1. Mosaiced image
2. Correlation file (optional)

9 Algorithm- Sky Images

This section describes how ommosaic works when the input images are all **OM** product sky-images.

STAGE 1- Checking of input files

```
Read in the list of input images from the parameter imagesets

Check that all the input files are either sky-images or unrotated images- if not stop the program wit

Check that all the files have different names- if not stop the program with an error message
```

STAGE 2- Some initialization- Inputting of various bits of information from the input files.

```
Loop over each input file

    1) Extract the values for the following FITS keywords in the file header
        1) FILTER   - the OM filter
        2) OBS_ID   - the observation identifier
        3) RA_PNT   - RA pointing direction of OM (degees)
        4) DEC_PNT  - Dec pointing direction of OM (degrees)
        5) POSCOROK - True if astrometry correction applied, FALSE otherwise
        6) RAOFFSET - Computed RA offset correction * cos(DEC_PNT) (arcsecs) - 0 if POSCOROK = FALSE
        7) DEOFFSET - Computed Dec offset correction (arcsecs)- 0 if POSCORK  = FALSE
        8) CDELT1   - Plate-scale in RA direction
        9) CDELT2   - Plate-scale in Dec direction
       10) CRPIX1   - Reference x pixel
       11) CRPIX2   - Reference y pixel
       12) CRVAL1   - RA value at CRPIX1
       13) CRVAL2   - Dec value at CRPIX2
       14) BINAX1   - Binning factor (0 or 1)
       15) BINBPE   - Binning on (True or False)
       16) WINDOWDX - Image x-axis length
       17) WINDOWDY - Image y-axis length
       18) DATE_OBS - Start time of observation
       19) DATE_END - End time of observation
       20) EXPOSURE - The exposure time (secs)

    2) Compute the following quantities
        1) RA of bottom left-hand corner of image
        2) Dec of bottom left-hand corner of image
        3) RA of bottom right-hand corner of image
        4) Dec of top left-hand corner of image
        5) Modified Julian date of mid-point of exposure

End loop
```

STAGE 3- Checking the astrometry of the images- Some images may have had an astrometry correction applied to their **CRVAL1** and **CRVAL2** values. For those that don't, a correction will be applied to **CRVAL1** and **CRVAL2** by interpolation/extrapolation using a list of the corrections applied to the rest and a list of their mid-point exposure times.

```
Create an empty list (raList) to store  computed astrometry RA  offsets
Create an empty list (decList) to store computed astrometry Dec offsets
Create an empty list (timeList) to store the computed mid-observation times


Loop over each input file

    If POSCOROK is true then:
        1) Add the RA astrometry correction to raList.
        2) Add the Dec astrometry correction to decList.
        3) Add the mid-point exposure time to timeList.
     End if
   End Loop

If the size of raList is zero then terminate STAGE 3 and go on to STAGE 4

Sort raList, decList and timeList into increasing time order


Loop over each input file
    If POSCOROK is false then:
        1) Compute an interpolated/extrapolated RA astrometry correction and replace its RAOFFSET va
        2) Compute an interpolated/extrapolated Dec astrometry correction and replace its DEOFFSET v
    End if
End Loop
```

STAGE 4- Adding extra astrometric keywords to the image files FITS headers to say whether or not an interpolated/extrapolated astrometric correction has been computed.

```
Loop over each input file
    If POSCOROK is TRUE then:
        1) Add a new boolean FITS keyword POSINTRP to the FITS header and set its value to FALSE.
        2) Add a new 32-bit float keyword RA_OFF to the FITS header and set its value to 0.
        3) Add a new 32-bit float keyword DEC_OFF to the FITS header and set its value to 0.
    else
        1) Add a new boolean FITS keyword POSINTRP to the FITS header and set its value to TRUE.
        2) Add a new 32-bit float keyword RA_OFF to the FITS header and set its value to the
           interpolated/extrapolated value.
        3) Add a new 32-bit float keyword DEC_OFF to the FITS header and set its value to the
           interpolated/extrapolated value.
    end if

End Loop
```

STAGE 5 - Alignment of images by cross-correlation **(This stage is optional and in the omichain will be skipped)**

Read in the value of the input parameter **mincorr**. If its absolute value is less than 0.0001 then skip this stage and go on to **STAGE 6**. Otherwise, please refer to **Section Image Alignment**.

STAGE 6 - Creation of the output image, exposure-image and quality-image.

```
1)  Find the maximum left-hand corner RA coordinate (RAL) from all the images
    (remember that the RA increase leftwards)
2)  Find the minimun right-hand corner RA coordinate (RAR) from all the images
3)  Find the minumum bottom edge Dec coordinate (DECL) from all the images
4)  Find the maximum top edge Dec coordinate (DECU) from all the images
5)  Determine the number of pixels in the x-direction (nx) using the plate-scale
6)  If nx exceeds 5000 (possible if using images from different observations),
    give a warning message and reset nx to 5000.
7)  Determine the number of pixels in the y-direction (ny) using the plate-scale
8)  If ny exceeds 5000 (possible if using images from different observations),
    give a warning message and reset ny to 5000.
9)  Create a 2-d 32-bit float image to store the stacked PRIMARY image.
10) Create a 2-d 32-bit float image to store the computed EXPOSURE image.
11) Create a 2-d 16-bit integer image to store the computed QUALITY image.
12) Set CRPIX1 to nx / 2 and CRPIX2 to ny / 2 (these refer to the output image)
13) Set CRVAL1 to (RAL+RAR) / 2 and CRVAL2 to (DECL+DECU)/2 (these refer to the
    output images)
```

STAGE 7 - Set the pixels in the output **PRIMARY**, **EXPOSURE** and **QUALITY** images.

```
1) If the images are comprised of a mixture of low/high resolution images set a variable mixedImages to
true- otherwise set it to false.

2)  Loop over input image files

    Load the image from the PRIMARY block in the file.
    If mixedImages=true and the image is a high-resolution one, then multiply each pixel of the image
    by four (to effectively convert it into a low-resolution one for interpolation purposes).


     Loop over pixels in output image
        1) Convert the tangent-plane coordinates, x and y, of the pixel to RA and Dec.
        2) Convert RA and Dec to input image tangent-plane coordinates, x1 and y1, using the WCS values
           for the image.
        3) Check to see that x1, y1 lies within the image, if not move onto the next pixel.
        4) Compute the value of the image at x1, y1 by 2-d bilinear interpolation and add this
           value to the output image pixel value at x, y.
        5) Compute the value of the exposure image at x1, y1 by 2-d bilinear interpolation and add
           this value to the output exposure image pixel value at x, y.
```

```
       6) Compute the value of the quality image at x1, y1 by determining which bits are set at
          each of the 9 neighbouring pixels and then by setting the corresponding bits of the quality-i
          pixels at x, y.

    End loop
  End loop


3)  Divide each pixel value of the output PRIMARY image by its corresponding EXPOSURE pixel value.
```

STAGE 8 - Creation of the mosaiced image file.

Determine the name of the output image file from the parameter **MOSAICEDSET** Create a **FITS** file with that name.

Add various keywords to the FITS header, including the following-

1. **FILTER** - the OM filter (could be more than one)
2. **OBS_ID** - the observation identifier (could be more than one)
3. **RA_PNT** - Aveaged RA pointing direction of OM (degrees)
4. **DEC_PNT** - Averaged Dec pointing direction of OM (degrees)
5. **CDELT1** - Plate-scale in RA direction
6. **CDELT2** - Plate-scale in Dec direction
7. **CRPIX1** - Reference x pixel
8. **CRPIX2** - Reference y pixel
9. **CRVAL1** - RA value at **CRPIX1**
10. **CRVAL2** - Dec value at **CRPIX2**
11. **BINAX1** - Binning factor (0 or 1)
12. **BINBPE** - Binning on (True or False)
13. **WINDOWDX** - Image x-axis length
14. **WINDOWDY** - Image y-axis length
15. **DATE_OBS** - Minimum **DATE_OBS** of all the input images **DATE_OBS**
16. **DATE_END** - Maximum **DATE_END** of all the input images **DATE_END**
17. **POSCOROK** - The input images **CRVAL1** and **CRVAL2** coordinates were astrometrically corrected (some maybe by interpolation or extrapolation)
18. **RAOFFSET** - The mean **RAOFFSET** value from all the input images.
19. **DEOFFSET** - The mean **DEOFFSET** value from all the input images.

```
Add the PRIMARY image to the output file.
Add the EXPOSURE image to the output file.
Add the QUALITY image to the output file.
```

9.1 Coordinate conversions

1. Conversion from tangent-plane to celestial coordinates, and the reverse, is done using routines from the SLALIB Positional Astronomy Library 2.5-3

9.2 Initialization (image alignment mode)

```
Read the list of input files on the command-line.


Read the input parameter nsigma
Read the input parameter minnumpixels
Read the input parameter minfraction

Set nFiles to the number of files


Loop through each file (i=1 to nFiles)


    Compute the mean  background of the image.
    Compute the number of image pixels that have a value >= mean background +
    nsigma * sqrt(mean background)
    and store this value in the vector numberGoodPixels

End Loop

Create an empty vector, called alignment, that will store details of pairs of
images that can possibly aligned
Loop through each file (i=0 to nFiles-1)
    if numberGoodPixels[i] < minnumpixels skip this value of i
    Loop through from j=i+1 to nFiles - 1
        if numberGoodPixels[j] < minnumpixels skip this value of j
        if( the area overlap of the two images divided by the maximum area of the
        two images is less than minfraction skip this value of j
        Add i, j, and the number of usable pixels to the alignment vector.
    End loop
End loop

Sort the alignment vector in order of decreasing overlap area
```

9.2.1 Initialization example

```
Assume that there are 5 images and that after computing the overlap area
fractions the alignment vector stores the following data

    image index i image index j   overlap fraction

1)       1         2              0.1
```

```
 2)       1         3              0.1
 3)       1         4              0.1
 4)       1         5              0.5
 5)       2         3              0.1
 6)       2         4              0.6
 7)       2         5              0.1
 8)       3         4              0.7
 9)       3         5              0.6
10)       4         5              0.9


After sorting into decreasing overlap fraction by Alignment class (Second stage)

   image index i image index j   overlap fraction


 1)        4         5              0.9
 2)        3         4              0.7
 3)        2         4              0.6
 4)        3         5              0.6
 5)        1         5              0.5
 6)        1         2              0.1
 7)        1         3              0.1
 8)        1         4              0.1
 9)        2         3              0.1
10)        2         5              0.1


This will then give the order in which two images will be attempted to be aligned.
```

9.3 Image alignment

Two images are "aligned" by computing the cross-correlation coefficient **R** between them (please see section 11.5) for a range of x and y offset values that are added to the reference pixel coordinates of the second image. The x and y offsets that give the maximum **R** are subtracted from the reference pixel coordinates of the second image to **align** it with the first.

In the case of sky-images, the reference pixel coordinates is that given in the image file FITS header by the keywords **CRPIX1** and **CRPIX2**.

In the case of unrotated images, the reference pixel coordinates is that given in the image file FITS header by the keywords **WINDOWX0)** and **WINDOWY0**.

The algorithm works as follows

```
Read the input parameter mincorr (the minimum correlation coefficient for a
successful correlation)

Take the absolute value of mincorr.
Find the size (n) of the vector of alignment paramters (see previous section)
Record that image i index of the value stored in alignment[0] has been aligned
```

```
(this will be the reference image).

Loop from iter=1 to no upper bound
    Loop through from k=1 to k=n
        Obtain the image file indices i and j from aligment[k]
        If neither image i or image j have been aligned skip this value of k
        If both image i and image j have been aligned skip this value of k
        If(image i has been aligned then cross-correlate image j with image i,
        otherwise cross-correlate image i with image j
        If the computed cross-correlation coefficient is at least equal to mincorr
        then record that these two images have been aligned, store the correlation
        coefficient and the x and y shifts between the two.
        Otherwise record that these two images cannot be aligned.

    End loop
    If all the images have been aligned or none were aligned during the current
    value of iter, exit the iter loop

End Loop
```

9.3.1 MAD test for data outliers

The MAD (median-absolute-deviation) test works as follows:

1. Compute the median value of the data values
2. For each data value compute its absolute deviation from the median
3. Compute the median of all the absolute deviations (the MAD value)
4. choose a value for nsigma
5. For each data value divide its computed deviation by the MAD value
6. If this value exceeds nsigma conclude that this is a deviant point

9.3.2 Alignment example

Please refer to section **Initializetion example**

```
Create a vector of booleans, isStacked,  of size equal to the number
of images, and initialize to false.

 set isStacked[3] to true- this will be the reference image. (NB indices
start at zero).

1) Attempt to align image 5 with image 4
        If successful - set isStacked[4] to true

2) Attempt to align image 3 with image 4
        If successful - set isStacked[2] to true
```

```
3) Attempt to align image 2 with image 4
        If successful - set isStacked[1] to true

4) If image 5 has been aligned then
   attempt to align image 3 with image 5
   If successful - set isStacked[2] to true

5) If image 5 has been aligned then
   attempt to align image 1 with image 5
   If successful - set isStacked[0] to true

6) If image 1 has been, but image 2 has not been aligned then
   attempt to align image 2 with image 1
   If successful - set isStacked[1] to true
```

9.4 Cross-correlation

The cross-correlation coefficient, $\mathbf{R}(\Delta x, \Delta y)$ between 2 images, $image_1$ *and* $image_2$, with offsets Δx *and* Δy between their x and y origins, respectively, is computed from

$\mathbf{R}(\Delta x, \Delta y) = \frac{sum_1}{\sqrt{sum_2 \times sum_3}}$,

where $sum_1 = \sum_{i=1}^{nx} \sum_{j=1}^{ny} image_1(x_i, y_j) \times image_2(x_i + \Delta x, y_j + \Delta y)$,

$sum_2 = \sum_{i=1}^{nx} \sum_{j=1}^{ny} image_1(x_i, y_j) \times image_1(x_i, y_j)$,

$sum_3 = \sum_{i=1}^{nx} \sum_{j=1}^{ny} image_1(x_i, y_j) \times image_2(x_i + \Delta x, y_j + \Delta y)$,

where nx and ny are the image dimensions in the x and y directions, respectively. In the above formulae it is assumed that the mean image values have been subtracted from the images. **R** can vary from -1 to +1.

10 Stacking examples

10.1 Stacking of sky-images from differeent observations

Figure 1 shows the image from the stacking of each of the V-band full-frame sky images from 4 SA-95 observations. The command-line to produce the image was,

ommosaic imagesets="P0134921001OMS001FSIMAGV000.FIT P0134921101OMS001FSIMAGV000.FIT P0154150201OMS001FSIMAGV000.FIT P0410780201OMS008FSIMAGV000.FIT" mosaicedset=testimage.fits mincorr=0". Setting **mincorr** to 0.0 resulted in the images being stacked without being aligned. **Omdetect** was also run on the image, and the detected source-regions are shown overlaid on the image.

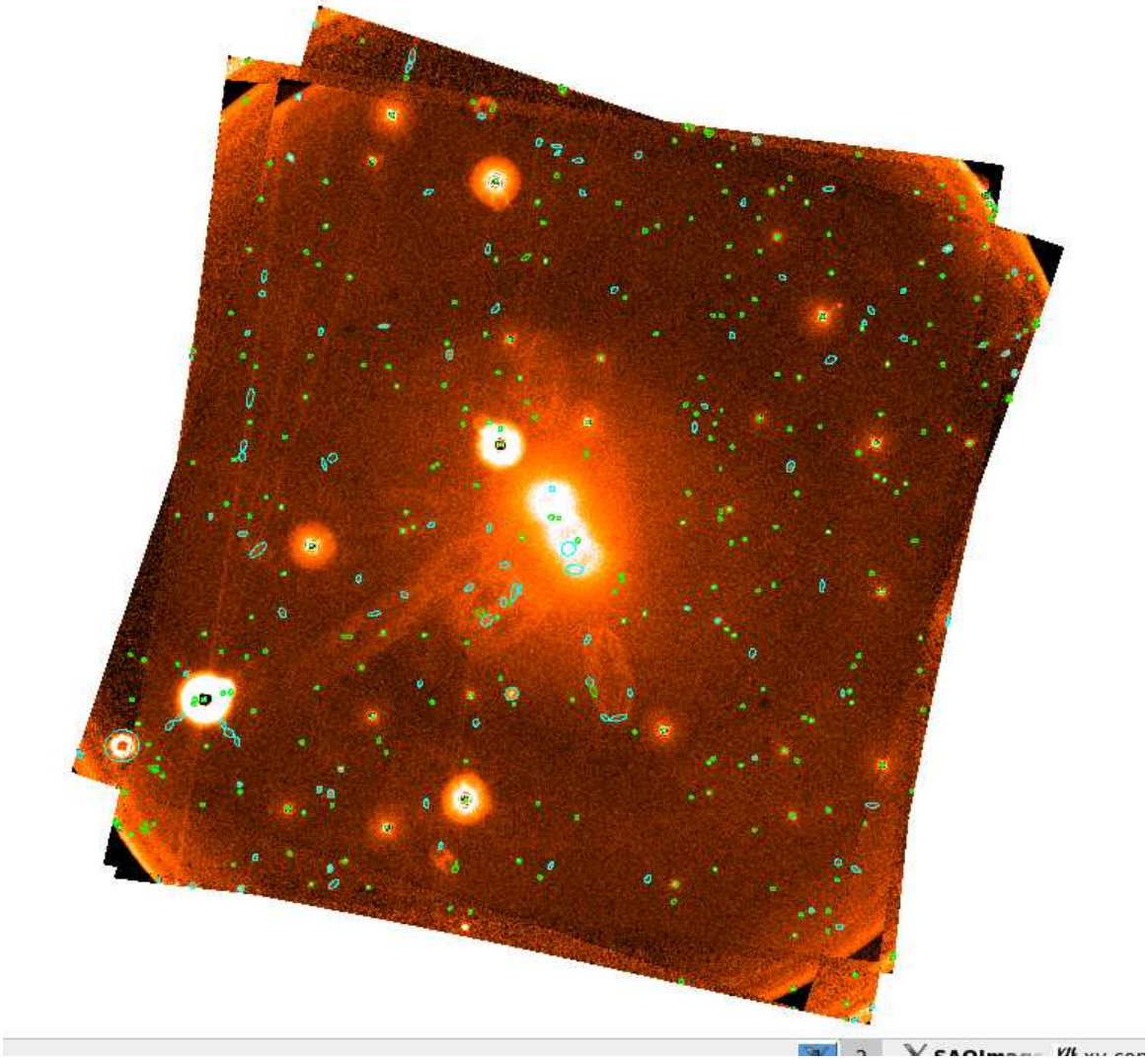

Figure 1: V-band sky-image produced from stacking the V-band full-frame sky-image from 4 SA-95 observations. The detected source regions are shown.

A careful inspection of the image shows the following features

1. Three read out streaks associated with the bright source near the lower left-hand side of the image, each one having a different orientation.

Figure 2 shows the stacked exposure image. The central region has an exposure about 5 times greater than that of a single image.

Figure 3 shows the stacked quality image. The central-enhancement regions (large circles near the centre), read-out streaks, bright-sources and edges are clearly seen.

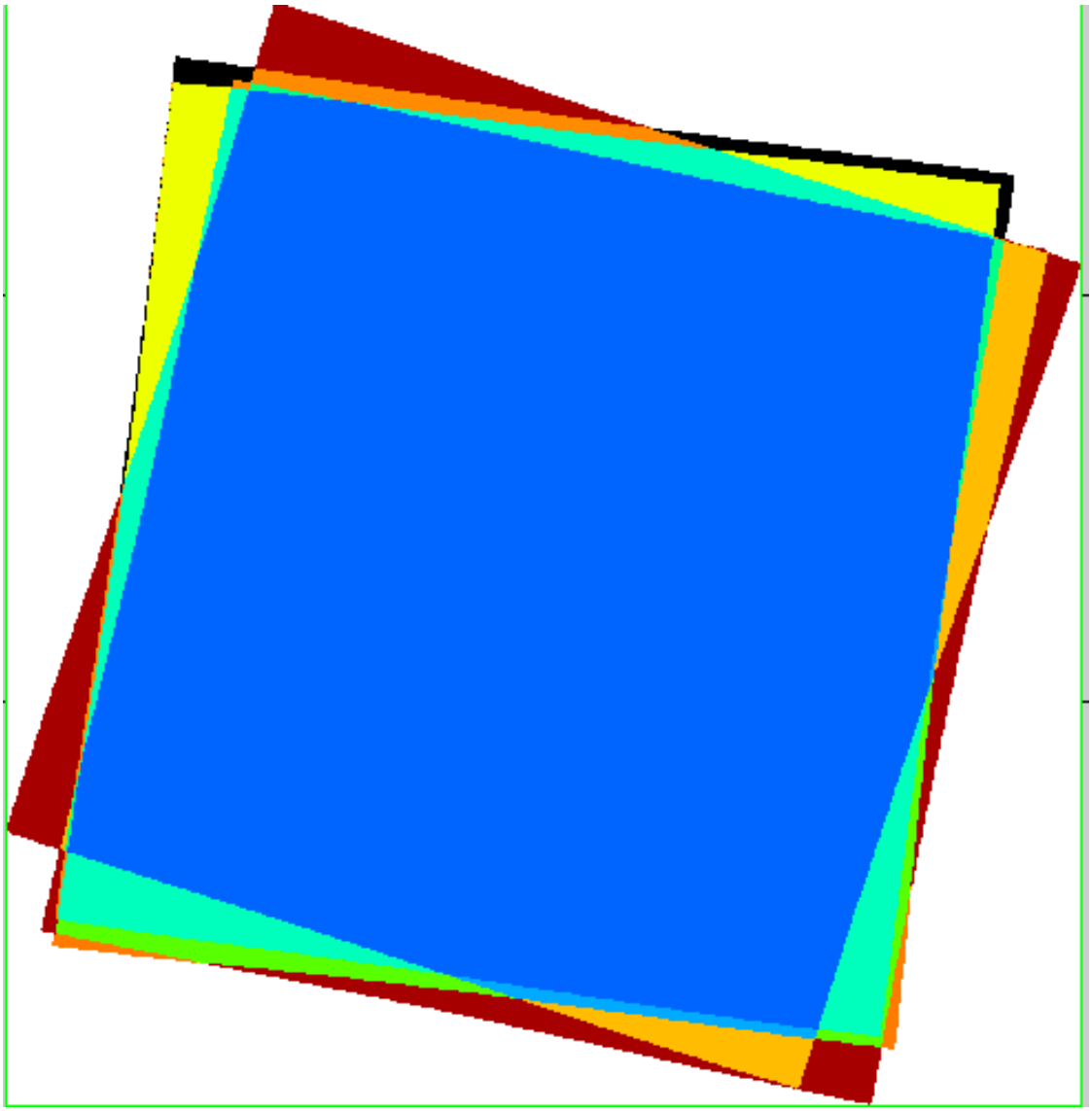

Figure 2: V-band exposure-image (seconds) produced from stacking the V-band full-frame sky-image from 4 SA-95 observations. The exposure varies from 1791 (light-brown) to 7224 (blue) and the area outside the image is white

10.2 Stacking of sky-images after internal image alignment

Before stacking images it is desirable to make sure that they are properly aligned first- by this it is meant that the absolute coordinate values (RA/DEC for sky-images) at the reference pixels are accurate. Any such errors will degrade the mosaiced image and possibly lead to source-detection and source-parameterization problems. To illustrate this, In Figures 4 and 5 we compare two mosaiced images from the stacking of the same two sky-image files, testimage1.fits and testimage2.fits. The latter file is a copy of testimage1.fits, but the reference pixels **CRPIX1** and **CRPIX2** have both had two pixels added. The image in Figure 4 has been created simply by stacking them, without alignment, using the command - **ommosaic imagesets="testimage1.fits testimage2.fits" mosaicedset=mosaicedimage1.fits mincorr=0"**. and the image in Figure 5 created using the command **ommosaic imagesets="testimage1.fits testimage2.fits" mosaicedset=mosaicedimage2.fits mincorr=0.5 correlset=correl.fits"**.

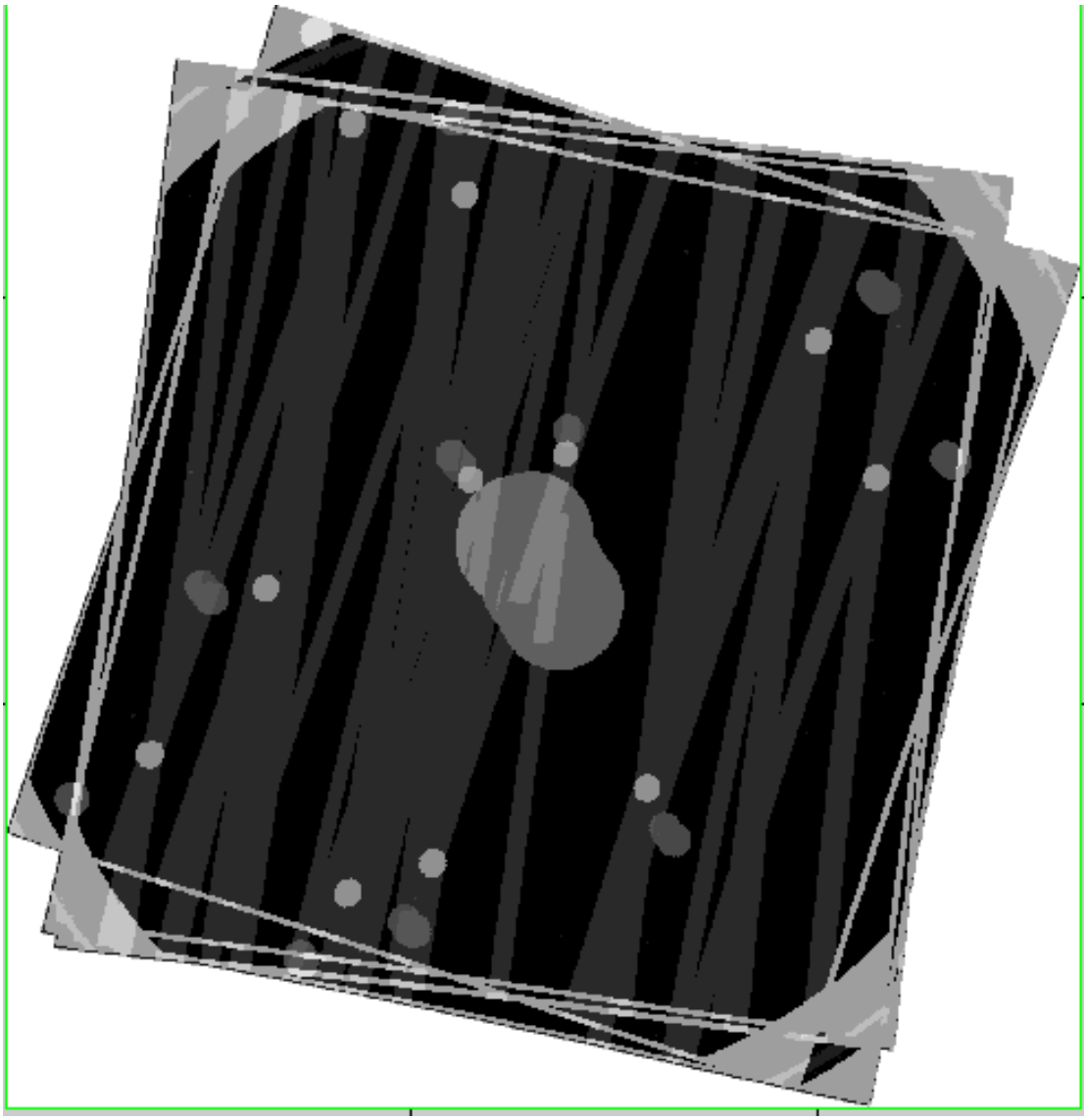

Figure 3: V-band quality-image produced from stacking the V-band full-frame sky-image from 4 SA-95 observations

For the latter mosaiced image, **ommosaic** correctly computed the reference pixel offsets (2 pixels) and corrected them internally before stacking the two images.

A comparison of the two figures shows that

1. The first image is obviously blurred compared to the second one.
2. Many of the sources classified as **extended (cyan region)** in the first image are classified as **point-like (green regions)** in the second one.

Figure 6 shows the correlation image produced by **ommosaic** and stored in the file **correl.fits**. The image is 21x21 pixels and each pixel is ten times smaller than a pixel in **testimage1.fits**. As can be seen from the image, there is a well defined maximum correlation near the centre of the image, and this is a good indication that the image-alignment algorithim worked well.

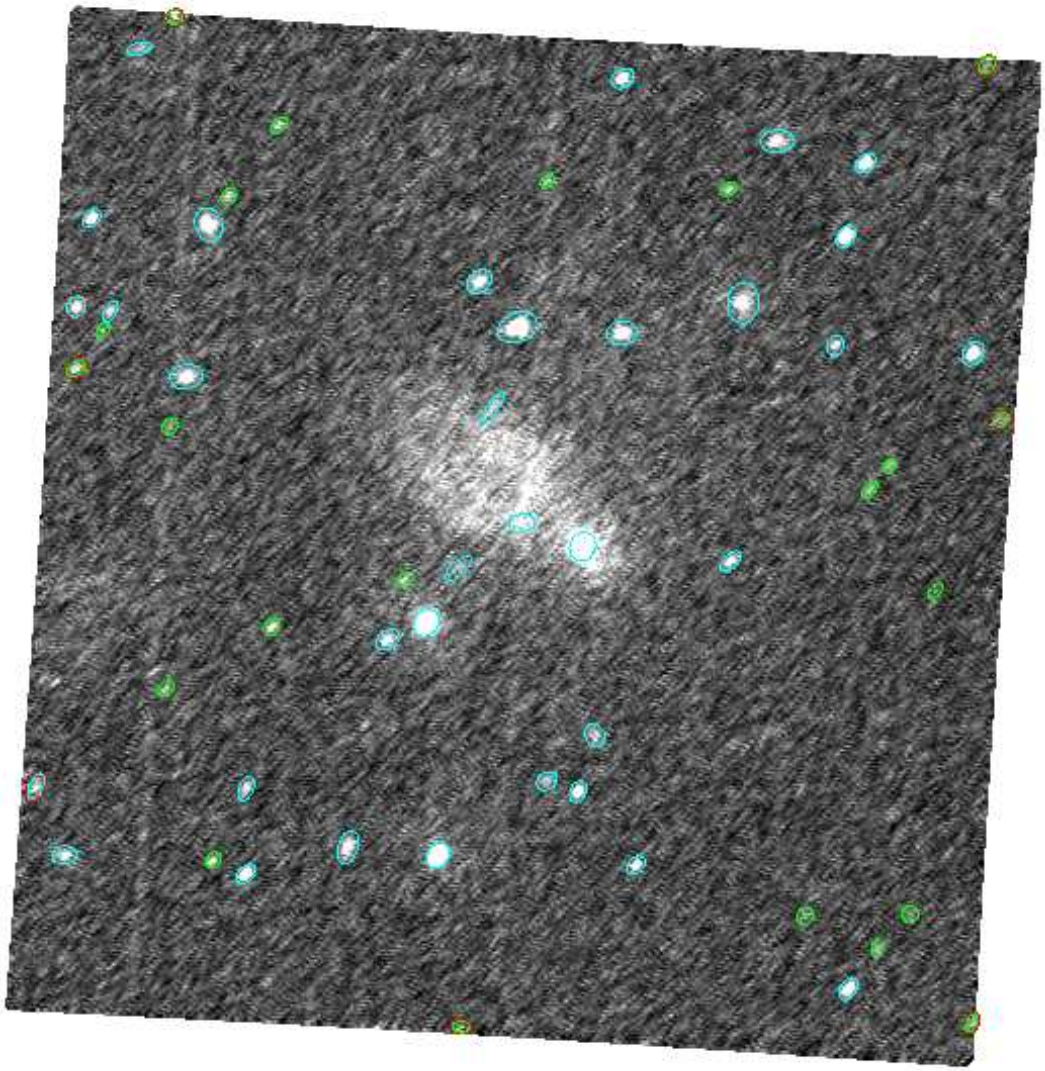

Figure 4: Mosaiced sky-image resulting from the stacking of two identical images with small reference pixel offsets **without image alignment**. The detected source regions are shown.

11 Testing

Ommosaic contains a test-harness that can check any of the following things:

1. That a list of sky images derived from a master image have been properly aligned and stacked.
2. That a list of unrotated images derived from a master image have been properly aligned and stacked.
3. That a list of sky-images derived from a master sky-image have been properly stacked.
4. That a list of unrotated images derived from a master sky image have been properly stacked.

Due to time constraints, only the two first of these tests are done when the package is built. The latter

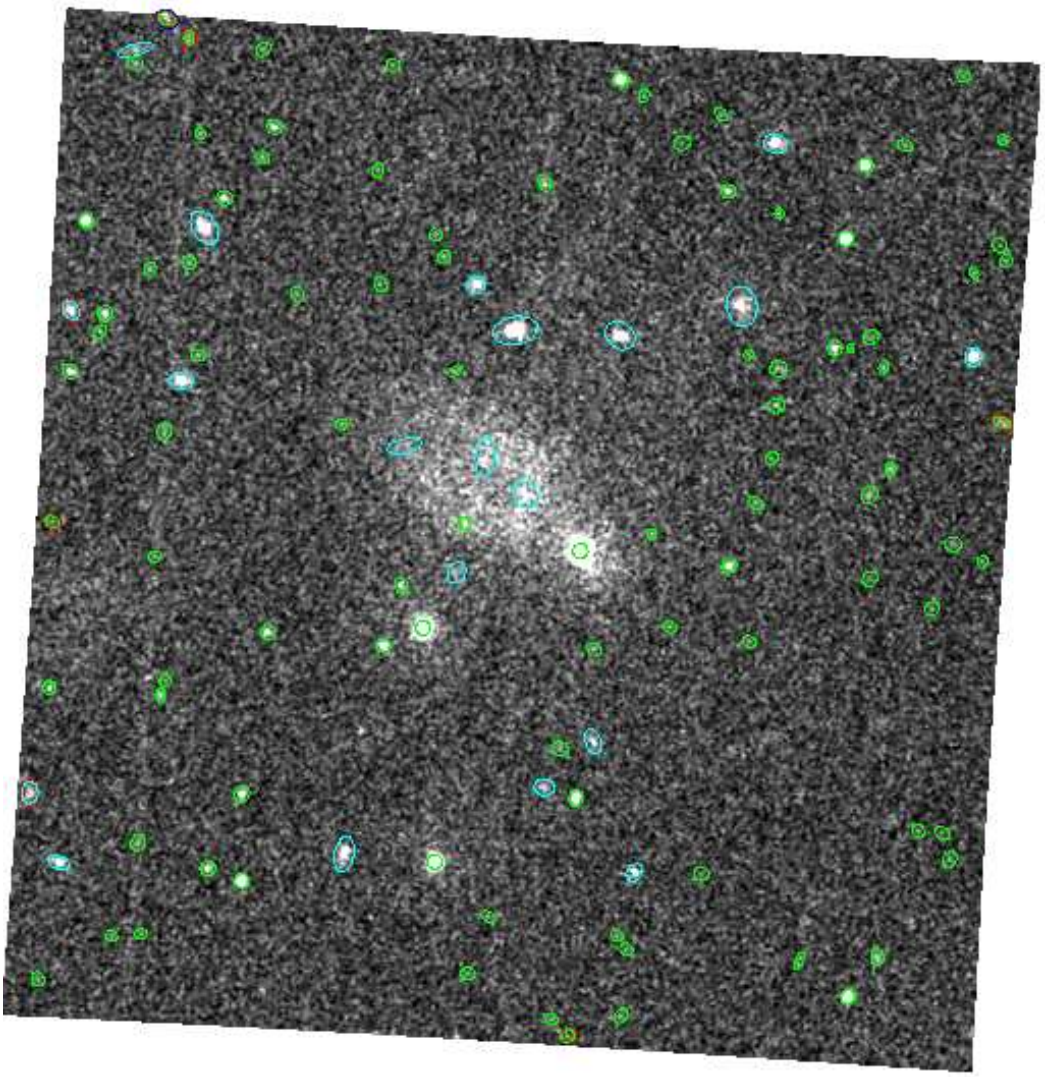

Figure 5: Mosaiced sky-image resulting from the stacking of two identical images with small reference pixel offsets **with image alignment**. The detected source regions are again shown.

two can be done by editing the program **createtest.cc** in the directory **test**.

A program **createst.cc** does the following

1. Creates a 256x256 test image file (**test.fits**), with a random background (1-500) and a random number of point-sources (10-100) at random positions on the image.
2. A master unrotated image file (**test1.fits**) is cloned from test.fits and FITS header keywords such as **WINDOWX0, WINDOWY0** are added to it.
3. 10 Unrotated-images files are cloned from **test1.fits**, and the FITS keywords **WINDOWX0** and **WINDOWY0** are given random small offsets.
4. A master sky image file (**test2.fits**) is cloned from **test.fits** and FITS header keywords such as **CRPIX, CRVAL1** are added to it.

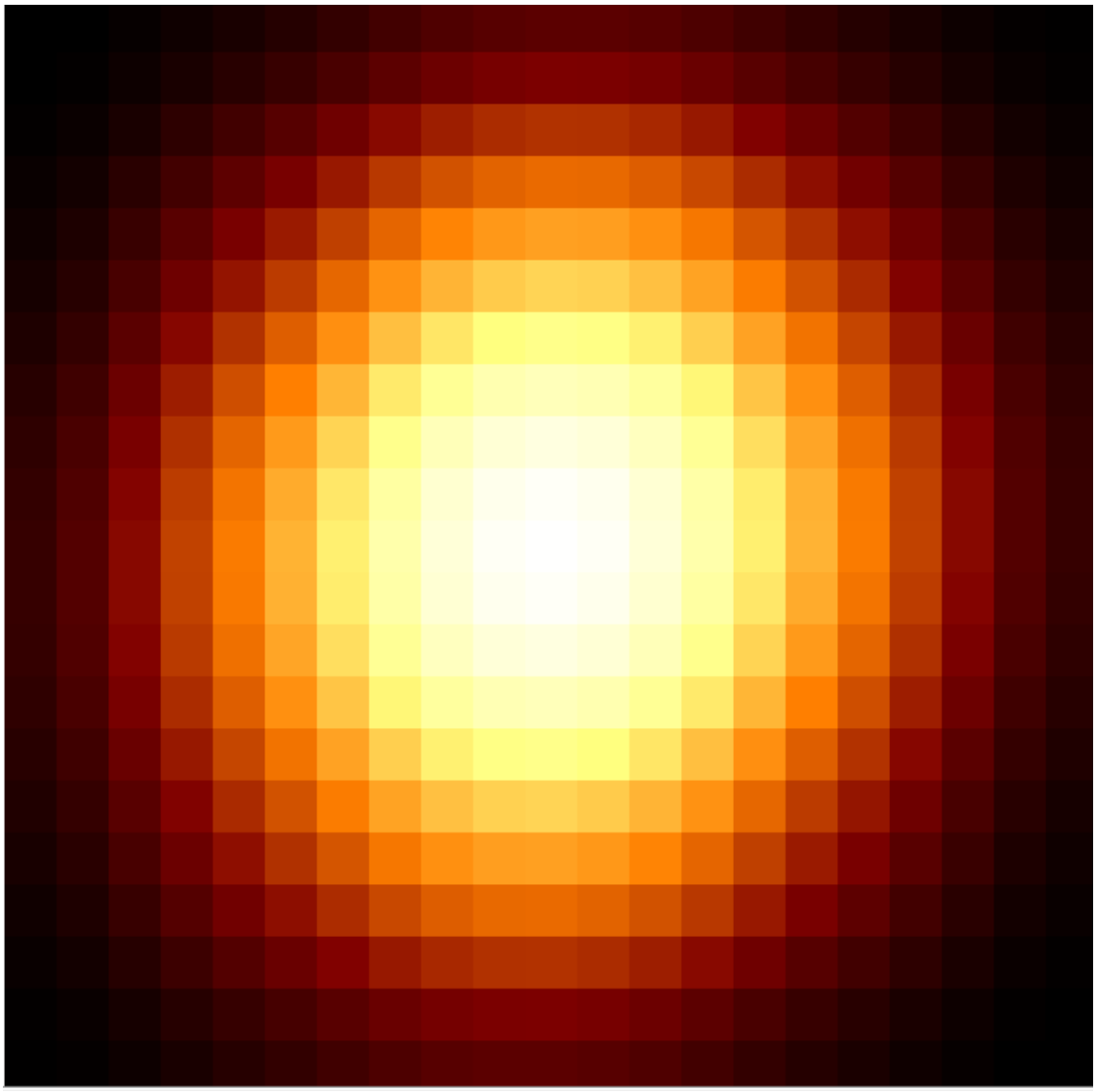

Figure 6: Correlation image produced by ommosaic

5. 10 Sky-images files are cloned from **test2.fits**, and the FITS keywords **CRPIX1** and **CRPIX2** are given random small offsets.
6. **Ommosaic** is run on **test1.fits** and **test2,fits** to produce output files **test3.fits** and **test4.fits**.
7. A program called **checkoutput.cc** compares **test3.fits** with **test1.fits**- it checks that the image dimensions are the same as well as each corresponding image pixels. Any differences (beyond rounding errors, etc) are reported and the test is deemed to have failed.
8. checkooutput.cc then compares **test4.fits** with **test2.fits**, and does the same tests as before.

For program development testing the number of images can be increased and the image dimensions, etc, changed.

12 Comments

- none

12.1 Future developments

- Whilst tests have so far shown that scattered-light features do not effect the cross-correlation algorithm, further testing is underway and the method should be used with caution.
- Tests using the cross-algorithm on aspect-corrected sky-images have so-far shown the computed offsets to be small (less than 0.2 pixels). If both the aspect-correction and the cross-correlation algorithm were perfect, they should be zero. However the error in the aspect-correction can be up to 1 arc sec, and further testing needs to be done to evaluate any differences.
- It would be desireable to perform a further aspect correction on the sky-images- this would require either a new **OM** task or, possibly, a modification to **omsrclistcomb**. Note that even if the sky-images have not been aspect-corrected, the coordinates of the sources in the observation source-list file may have.

References