Procedure for segmenting the image with “finder.py”

Carlos Brandt – chbrandt@lncc.br

The code “finder.py” was written with the purpose to segment (astronomical) images using two segmentation algorithm, and than, optionally, a positioning cross-correlation for segmented objects .vs. truth-table check.

The code has a main function called “run” that deals with the options for segmenting, cross-checking , cleaning and seeding points.

Is mandatory to run the code an image array (matrix with float values), the rest of the parameters are optional:

- thresh : Threshold (min) value for segmentation
- seeds : Seed point for specific segmentation
- truth : Truth table to match with segmented objects
- radius: Distance for tables (objects_vs_truth) matching
- border : Size of the border if/to eliminate objects in image border

The image used in this example is the one often used by Anupreeta, from the CFHT:

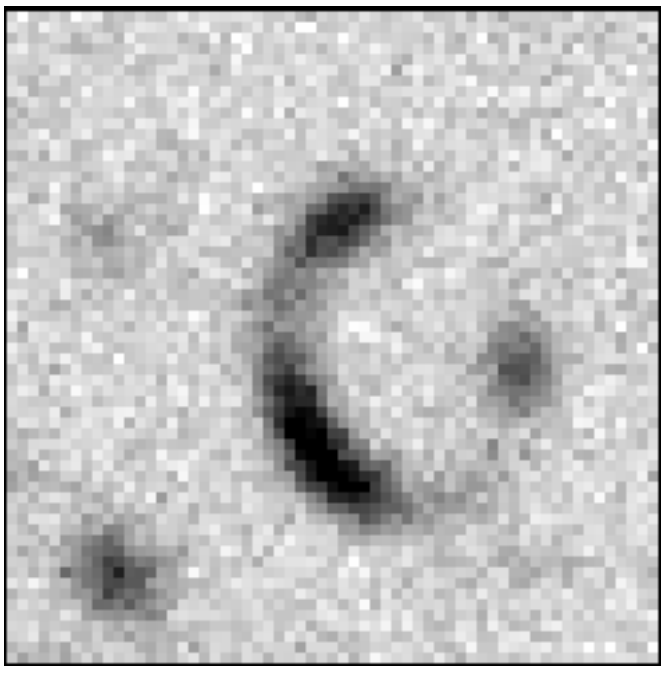

Figure 1: Arc from CFHT.

The procedure (pipeline) encoded in finder.py is meant to be a model algorithm for the purpose of segmenting + cross-checking the output and quality of the segmentation with a truth-table. It is not meant to be **the** (or **a**) finder, lots of details can be (and should be) improved.

This algorithm is to enlighten the work being done for the ArcFinders comparison article, with the experience I have with data/image handling.

The code “finder.py” implements the following methods/functions, which are used depending on the options given:

- Filtering:
  - First step, mandatory, is a Gaussian filtering with sigma=3 pixels
- Segmentation:
  - *Thresholding*: Objects (regions with pixels) above a threshold value and area bigger than 9 pixels are segmented. The segmented image is cleaned (before the minimum area (9) filter) with morphological operations (binary opening) for removal of spurious (noise) detections.
  - *Region-Growing*: If seeds are given, a region is grown around the seed till the “above threshold” condition is no more satisfied.
- Threshold value:
  - If the user doesn’t give a threshold value, one will be automatically calculated based on the (image) histogram’s maximum value (the *fashion* of intensities distribution) plus image’s standard-deviation. Which will set the threshold value (“thresh”) right above the background noise.
- Measurements:
  - Objects “center-of-mass” (hereafter centroids) are computed.
- Cross-correlate/check:
  - If truth-table points are given, a matching between “centroids” and “truth-points” is done. Completeness and contamination are also calculate.

The code, as you should be noticing, is flexible enough to be used for segment-only, 1st-moment stability and truth-table checking; The capabilities include the use of a floting (observed) image as well as a already segmented (binaryor labeled/uint) image.

Read further for the cases of use.

** To run the code, one needs python, numpy, scipy and matplotlib. logging is also needed since a logfile (“finder.log”) is within the output files.*

So… how to use it?

To use the program I’ve created a (tmp) directory where I put the code (finder.py) and the image I wanted to use:

- finder.py
- arc_AM.fits (figure 1)

And start the environment:

```
>>> import finder        # Remember that the main function is called “run()”
>>> import pyfits
```

>>> img = pyfits.getdata('arc_AM.fits')

(You will find all the information you need with the help() function)

>>> help(finder.run)

If none of the optional arguments is given, the algorithm runs the "thresholding" segmentation methods only, automatically choosing the "thresh" value, and outputs the segmented and smoothed version of input image. Also, an image with the detected centroids and the logfile (finder.log) are written. At the standard-output (screen) you will see the information regarding segmented object centroids and tables-matching (if applied):

>>> finder.run(img)

Below are the (3) images given as output from the code: (i) the segmented objects-image, (ii) the smoothed image, (iii) centroids location image.

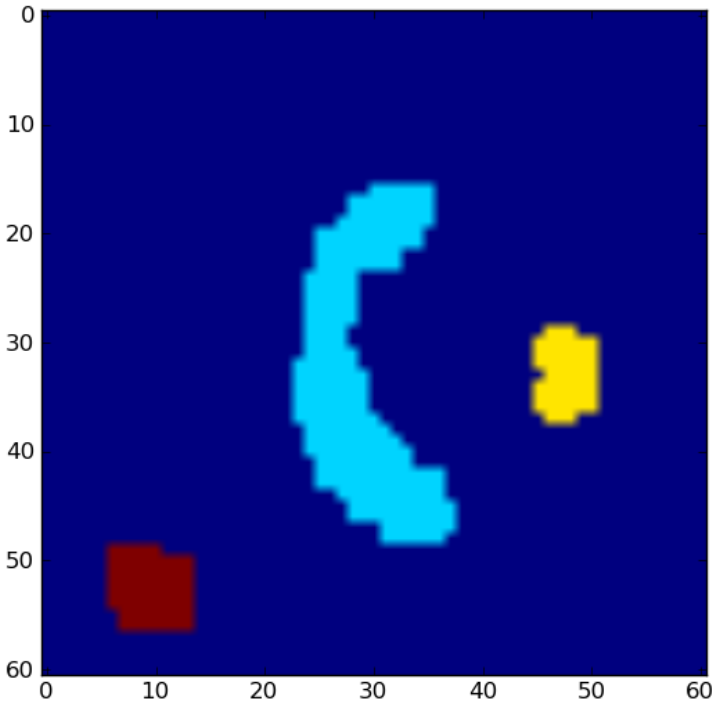

Figure 2: Threshold segmentation result. 3 objects found.

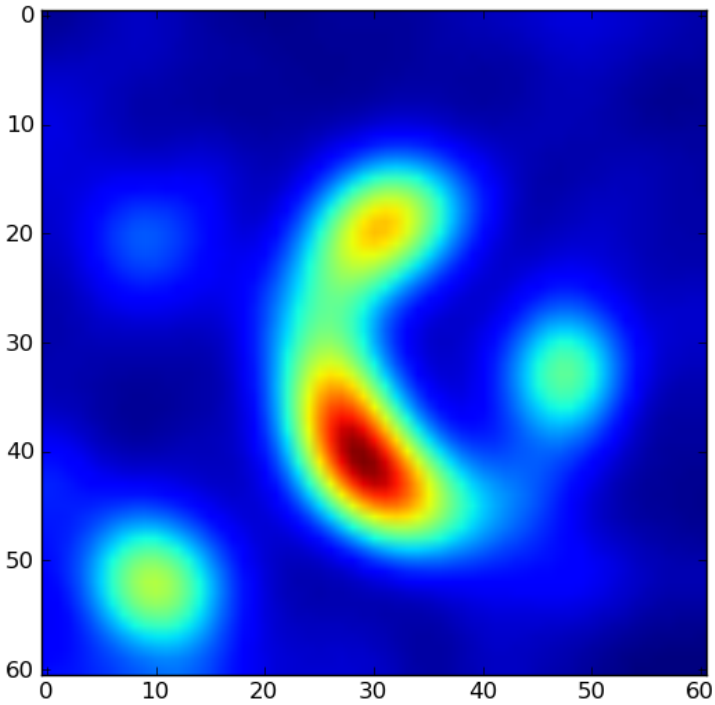

Figure 3: Gaussian smoothed with a square sigma window of size 3.

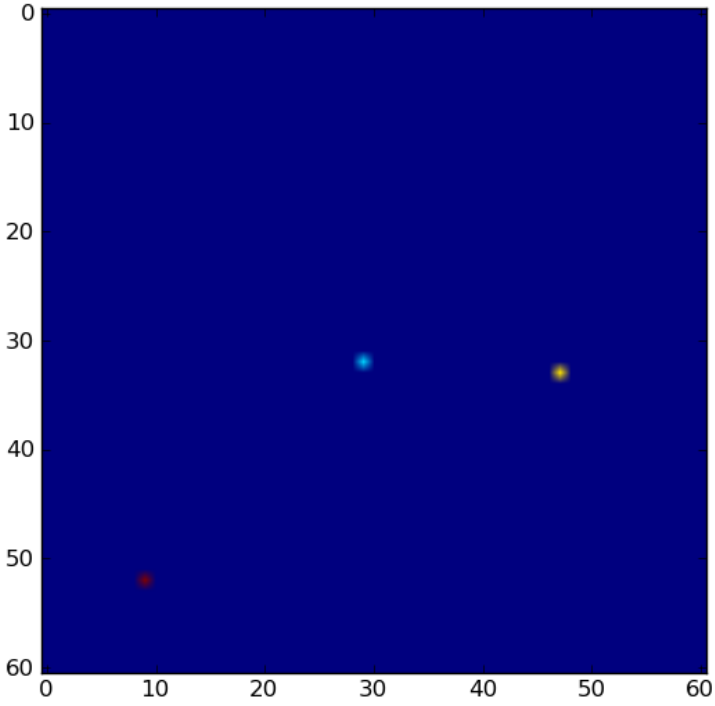

Figure 4: Found objects centroids (center-of-mass).

And, at the screen is given the following output info:

```
------------------------------
Segmented objects (centroid):
Xo Yo
29 32
47 33
9 52
---------
```

Now, suppose we have a seed point that localizes one/the object of interest:

- X = 30
- Y = 40

Then we would run the finder like:

```
>>> finder.run(img,seeds=[(30,40)])
------------------------------
Segmented objects (centroid):
Xo Yo
29 32
---------
```

Where the pair of values “29,32” is the centroid of the segmented object:

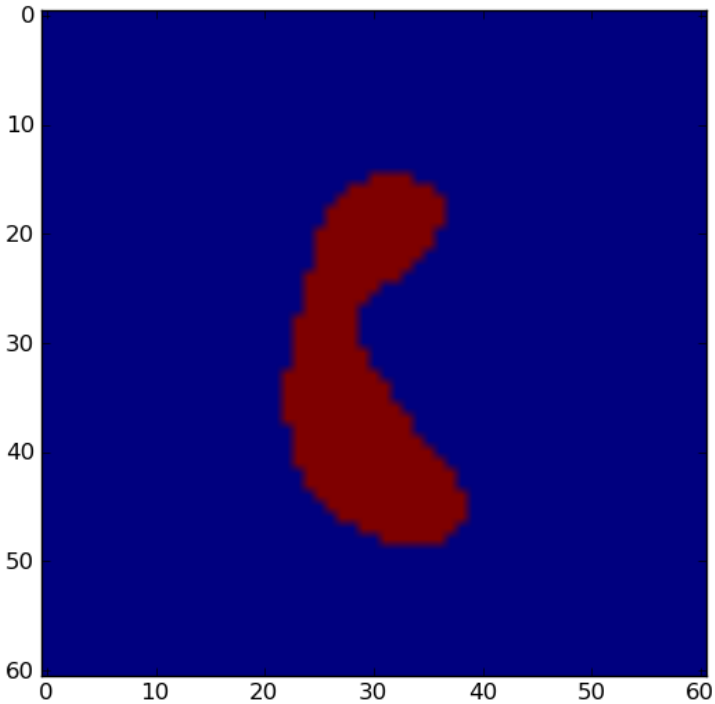

Figure 5: Segmentation (RG) around the seed point x=30 & y=40.

Now suppose you have a “truth” point and want to check if the thresholding + center-of-mass computing methods are capable of locating this “true” object:

```
>>> finder.run(img, truth=[(30,40)], radius=10)
------------------------------
Segmented objects (centroid):
Xo Yo
29 32
47 33
9 52
---------
------------------------------
Matched points (truth neibour):
Xo Yo neib? nearest_X nearest_Y
29 32 True 30 40
47 33 False 30 40
9 52 False 30 40
------------------------------
Completeness:  1.0
Contamination:  2.0
```

Then we have these extra information’s regarding the cross-checking of the tables: the one “truth” given as input and the one created (internal) by the segmentation. We are able to see that one of the objects were found and the other two did not matched with the truth-table.

* Default radius value is “10” pixels. See the functions help info for options.

* Contamination here if greater than “1” because the given truth-table is larger then the “sample” table.

Now, suppose we have some seeds and a truth point. I’ll use approximate values for seeds like we would if if had just a clue about the location. The truth points are known values:

```
>>> finder.run(img, seeds=[(30,40),(10,50)], truth=[(47,33),(29,32)])
------------------------------
Segmented objects (centroid):
Xo Yo
29 32
9 52
---------

------------------------------
```

Matched points (truth neibour):
Xo Yo neib? nearest_X nearest_Y
29 32 True 29 32
9 52 False 29 32

Completeness: 0.5
Contamination: 0.5

And we can see that it performs correctly with the matching; half of our points were known to match:

30,40 <-> 29,32

and half (10,50) would not match anything. Notice that the radius value used is the default one ("10" pixels).

Now, as I think is the interest of the people reading this document, suppose one already has a segmented image – a binary image (0,1) – output of any other program.

I have here (figure below) an output from Lenzen's arc-finder. Of course the same image (figure 1) was used.

Figure 6: Lenzen's arc-finder (segmented image) output.

And we want to check these objects with our own truth-table (which, is populated by one point, in here). Since this is a binary image, the "thresh" value of the finder I am giving should be set to something between (0:1).

```
>>> finder.run(simg,thresh=0.1,truth=[(47,33)])     # 'simg' is the lenzen's output
------------------------------
Segmented objects (centroid):
Xo Yo
9 20
29 20
47 32
28 39
9 52
---------
------------------------------
Matched points (truth neibour):
Xo Yo neib? nearest_X nearest_Y
9 20 False 47 33
29 20 False 47 33
47 32 True 47 33
28 39 False 47 33
9 52 False 47 33
------------------------------
Completeness:  1.0
Contamination:  4.0
```

Where we can see, as expected, one object was correctly found/matched and other 4 didn't. The segmented (labeled) image and their centroids for this particular run is:

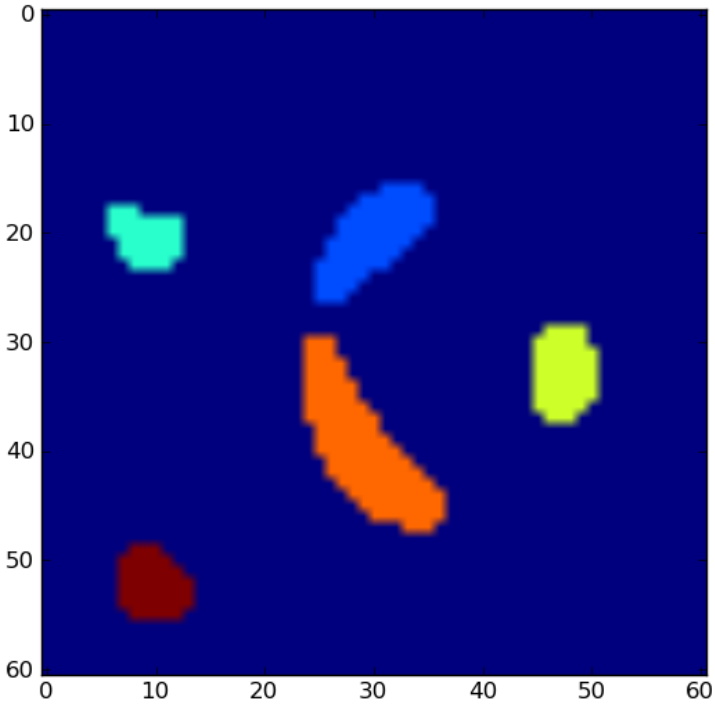

Figure 7: Segmented image by our "finder" for the binary input image.

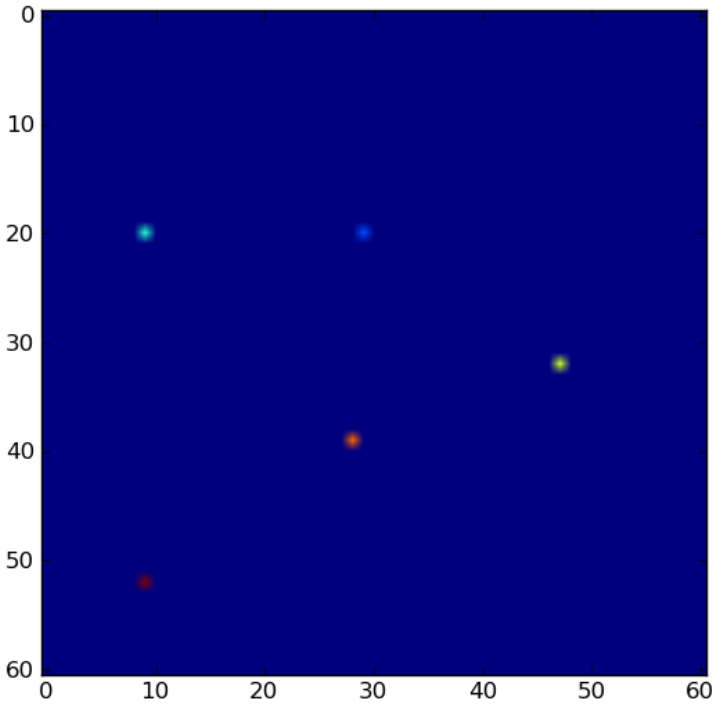

Figure 8: Centroids for the current example.

So, this is the basic algorithm for checking image segmentation and cross-correlate positional tables. It has basic segmentation methods, and simple parameters check. The minimum bias possible was added during the implementation so that the input can control well the pipeline and results are keep simple and clear.

Lots of details can be added/implemented, though; in particular, for the current purposes, I think the the "measurements" and "table-matching" (classification) can be improved to verify more features of the segmentation/detection. For instance, one can compute 2nd and 3rd moments, to estimate object's area, length-to-width and/or curvature.