university of groningen / faculty of mathematics and natural sciences / kapteyn astronomical institute

Bachelor Thesis Project

Separating SoFiA's Sources

An identification and separation module for SoFiA

M.J.F. Versteeg

supervised by
Nadine Giese
Thijs van der Hulst

July 12, 2016

Abstract

SoFiA (Source Finding Application) is a flexible piece of software that performs automated 3D detection and parametrization of sources in H I data cubes. Sometimes, however, the H I of galaxies that are close together might overlap. SoFiA is then unable to unambiguously assign the H I regions to a galaxy and thus falsely identifies the galaxies as one source. This paper presents a new module for SoFiA, capable of identifying ambiguously assigned areas and separating those, in 2D, into their respective components. By applying a watershed algorithm to a Gaussian-filtered moment-0 map of the data cube, an initial model is defined for each of the galaxies. These shapes are then dilated until the SoFiA mask is completely filled, thus separating the mask into its component pieces.

1 Introduction

Since the 1930s, radio astronomy has been used to discover and analyze the Universe in ways not possible through optical astronomy alone. Radio astronomy has given scientists measurements of the Cosmic Microwave Background (CMB), new types of objects previously unknown to exist, such as quasars, and much more. In this day and age, with bigger telescopes and telescope arrays, the amount of data, and thus information, increases as well.

With upcoming H I surveys like WALLABY (with ASKAP) (Duffy et al., 2012), and surveys with the new APERTIF system on the Westerbork Synthesis Radio Telescope (WSRT) (Verheijen et al., 2008), a demand for a thorough and flexible automated application for source finding has been created. SoFiA (Source Finding Application (Serra et al., 2015)), is a modular application that combines source-finding algorithms in such a way that the user can flexibly select what modules to use for their specific research. While SoFiA satisfies many of the demands astronomers have, it is not without flaw. When used to find sources in reduced radio astronomical spectral line data, SoFiA can wrongfully flag two or more separate galaxies as one source, due to an overlap in their H I distributions. Because this can lead to less reliable results, it is important to develop a module for SoFiA that can intercept false source identifications and correct the source finding by separating the source into its loose components- one for each individual galaxy.

2 Method

Processing the data cubes consist of two primary steps: identification and separation. To identify the content of the data cube, an automated search in NED (NASA/IPAC Extragalactic Database) [1] is performed to find the galaxies contained within the right ascension (RA), declination (Dec) and redshift (z) limits of the cube. If multiple galaxies share a source ID in the mask cube (that has been created by SoFiA), an ambiguously assigned source has been found. In such cases, the second step of the process, separation, is required. A two-dimensional model of the part of the sky where the cube is located is created, using a Gaussian-filtered moment-0 map of the galaxy, to which a watershed algorithm is applied to create models of the galaxies according to their gas distribution. The models of the galaxies are then 'grown' using dilation to fit the original mask as created by SoFiA, ensuring that the original mask is divided between the individual sources.

In order to perform the above process, a module has been developed that can be added to the existing SoFiA program. The source identification and separation module has been developed in the Python programming language by the author and makes use of several packages, including NumPy (Van Der Walt et al., 2011), SciPy (Jones et al., 2001–), Astropy (The Astropy Collaboration et al., 2013) and Astroquery (Ginsburg et al., 2013). In order to create the images, MatPLotLib (Hunter, 2007) has been used. Ancillary data (See Appendix A) has been used in the development of this module. The source code identification and separation module can be found in Appendix D and online at the author's website.[2] At the time of writing, the identification and separation module requires the user to manually provide the cube, mask cube and moment-0 image. If this module gets implemented into SoFiA, however, no user interaction will be required and the module will operate autonomously. The identification and separation process as performed by the module will be discussed in more detail in sections 3 and 4.

NED has been chosen as the primary database because of its large amount of easily accessible information and easy-to-use web interface for comparison to the results of the identification to ensure its proper workings. If necessary, however, any other database accessible through the Astroquery module could be used by modifying the module.

The module is built up in such a way that it will accept data cubes of all sizes. Its coordinates can either be in the B1950 or J2000 epoch, as an automated check is built in. It compares the galaxies

[1] The NASA/IPAC Extragalactic Database (NED) is operated by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

[2] `www.astro.rug.nl/~versteeg/bscthesis/index.html`

NED finds within the cube's limits to the mask of sources that have been identified by SoFiA. A system message will inform the user of the galaxies found in their cube. The module will then check for separate galaxies that have been assigned the same source ID. If multiple galaxies share a source ID, it will print a warning, alerting the user of the shared source IDs.

3 Identification

Automated search query

Using the headers of the data cube, the module can determine the size of the cube. By using the RA and Dec of the center pixel and the coordinates of the corners to find the radius necessary to cover the entire cube, the region in which NED must be queried can be found. Using Astroquery, a NED search can then be conducted: Astroquery returns not only a list of objects, but also their basic data, including but not limited to coordinates (in RA and Dec), redshift and type of object. The module filters the resulting list using two criteria: firstly, the redshift limits of the cube itself: the frequency limits along the line of sight are also saved in the header data and can be used to determine a maximum and minimum redshift. Secondly, the module filters by type of object, set to galaxy ('G'). Although it can be set to another type or even to filter by a 'G' in any part of the type name, it is advisable to leave it as is. Filtering by type 'G' ensures that the list does not contain any duplicate entries. Since the NED query also returns pairs of objects, groups of objects etc., the list might not only contain the galaxy the user is looking for, but also the groups/pairs/cluster it could be a part of.

Mask comparison

Using the basic data of the galaxies, the RA/Dec coordinates obtained from NED of each galaxy are converted to pixel locations within the cube. The pixel values of the locations of the galaxies inside the data cube are then used to access the mask cube at that value along the line of sight. The mask cube is a 3D array filled with zeros, except at the location of the sources, where it will contain the source ID values. If a source ID is found along the line of sight at the location of the galaxy, the name, frame number and source ID are added to a list. The program generates such a list for each individual galaxy. The lists of galaxies can then be compared: if the names of two galaxies do not match and their source IDs *do* match, SoFiA has wrongfully assigned a single source ID to multiple galaxies. An automatically generated warning containing the names of the galaxies and the source ID is printed to inform the user of the error.

4 Separation

Model array creation

The source separation starts by creating a 2D model of the sky that has the same dimensions as the cube in RA and Dec. This model is based on the moment-0 image created by SoFiA. A Gaussian filter (with a variable standard deviation σ) is applied tot the moment-0 image. Initially, the sigma for the Gaussian filter is set to a minimum of $\sigma = 2$. The watershed algorithm can be then be applied. This is an automated process that removes progressively higher values from the moment-0 image, until the number of objects is equal to the number of galaxies that share a source ID. If the watershed does not result in the desired number of objects, the σ of the Gaussian filter is increased by 0.1 and the watershed is repeated until the correct number of objects is left. The module stops trying to create a model if σ reaches a limit, currently set at $\sigma = 15$.

The next step is to assign the proper identity to the created objects. This is necessary for the next step (dilation), in which the models are grown to fit the mask. In order to do this, the program compares the optical centers of the galaxies (from the NED query) to the centers of mass of each object and assigns identities via a simple distance comparison. In some cases, this will lead to an object being doubly assigned. The watershed is then deemed unsuccessful.

It is important to note that even with a variable standard deviation, not all moment-0 maps can be watershed to a correct model. If the program cannot watershed the filtered moment-0 map to the desired amount of objects or if it cannot properly assign identities to the objects, the watershed model creation is deemed impossible. In this case, the program will automatically switch to an alternative way of model creation: using diameter data from a NED query to create elliptical models of the galaxies. For the galaxies for which NED does not provide diameter data, generic circular models are created. If the models overlap, the module cannot separate the mask into individual components. If this happens, the model is recreated using objects that are half the original size, after which another attempt at separation will be made.

2D source separation

In order to perform the separation in 2 dimensions, the mask cube that has been created by SoFiA, is reduced to a 2D array by adding all frames along the line of sight. Within the model containing the ellipses (the overlay), the galaxy arrays are dilated using SciPy's built-in dilate function (see appendix C). Each galaxy array is dilated individually by an elliptical kernel with a size dependent on the size of the galaxy in question and its gas distribution. This information is stored in the moment-0 map. This method ensures that galaxies that contain more gas 'grow' faster than the ones that contain less gas. Furthermore, comparison to the moment-0 image allows for a more accurate separation: as the average amount of gas around a galaxy decreases, the dilation footprint reduces to a minimum size as well.

It is important to note that NED does not provide diameter data for all objects. For objects that have no diameter data available, the module creates a generic circular kernel, currently set to $r = \sqrt{5}$ pixels.

For optimization reasons, a subarray that encloses the 2D mask is defined to create boundaries in which the dilation takes place. The dilation fills up the subarray, ensuring that the galaxies do not overlap: only elements that have not yet been assigned a value in the subarray are added to the newly dilated ellipse. After every dilation step, the array (that now contains a region of only non-zero values, one value for each galaxy) is multiplied with the 2D mask (that contains only ones where the mask is defined and zeros elsewhere), resulting in a mask that is divided between the galaxies in question.

5 Results

In order to demonstrate the workings of the module, a selection of three cubes has been made. These cubes are part of the WHISP survey (van der Hulst et al., 2001) and are publicly available at WoW (Westerbork on the Web) [3]. The masks as well as the moment-0 images that are used have been generated by SoFiA, publicly available at GitHub [4].

UGC 1256

This cube contains not only UGC 1256, but also its companion UGC 1249. When this cube is used as input for the identification and separation module, the user will be alerted of a shared source ID:

```
Galaxies found in your cube: ['IC 1727', 'NGC 0672']
WARNING: SoFiA has given both IC 1727 and NGC 0672 the same sourceID:
    [1.0]
```

SoFiA has assigned source ID 1 to two galaxies, IC 1727 (UGC 1249) and NGC 0672 (UGC 1256). This means that the module will attempt to separate the 2D mask into two components. Firstly,

[3] http://wow.astron.nl/
[4] https://github.com/SoFiA-Admin/SoFiA

the module creates a model, using the Gaussian-filtered watershed moment 0 image as a basis, as shown in figure 1.

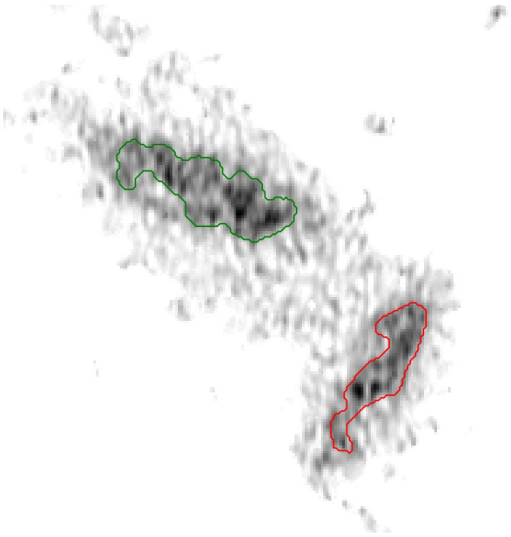

Figure 1 – Contours of models superimposed on the moment 0 image. The green contour represents the model for NGC 0672 (UGC 1256) and the red contour corresponds to the model for IC 1727 (UGC 1249)

This model is then used as a basis for the dilation, the result of which can be seen in figure 2:

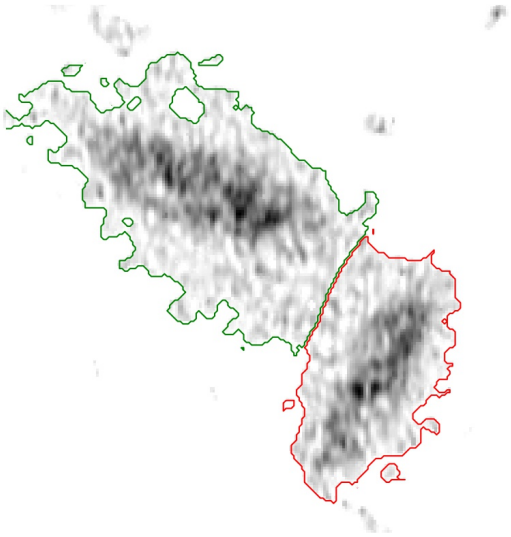

Figure 2 – 2D separated mask array superimposed on the moment-0 image.

The identification and separation module has been able to separate the mask into two components, one for each galaxy. Despite being in 2D, this separation can serve as a basis for further analysis.

UGC 2941

The process is not always as successful as demonstrated on cube UGC 1256. In some cases, the separation will not work properly. In order to demonstrate this, the data cube for UGC 2941 is used as input for the module. This cube contains not only UGC 2941, but also UGC 2492 and UGC 2943. While UGC 2941 is given its own source ID, UGC 2942 and UGC 2943 share one:

```
Galaxies found in your cube: ['CGCG 487-013', 'IC 0357', 'UGC 02942', '
    UGC 02943', 'MCG +04-10-020']
WARNING: SoFiA has given both UGC 02942 and UGC 02943 the same sourceID
    : [2.0]
```

which means the separation component will attempt to do its job. In this case, however, the watershed algorithm has been unable to create the right number of models. It thus switches to NED diameter data to use elliptical models of the galaxies. The NED diameter data, however, produces two ellipses which already overlap. (shown in figure 3). Since the module cannot separate galaxies of which the models already overlap, the program attempts to create the model again, using NED data, but shrinking the ellipses down to half size, leading to figure 4 (both cropped for clarity).

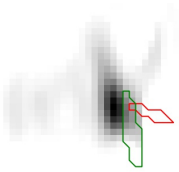

Figure 3 – NED model contours superimposed on the moment-0 image

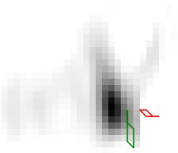

Figure 4 – Half-size NED model contours superimposed on the moment-0 image

Using the smaller model, the program is capable of dilating the image in order to fill up the mask. This, however, does not lead to a desired result, as shown in figure 5:

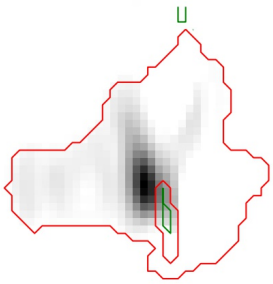

Figure 5 – 2D separated mask array superposed on the moment-0 image.

This weird dilation is most likely caused by a very small model having a very high average amount of gas (leading to a large footprint and thus fast growth, even if the area of one galaxy will thus 'invade' the area of the other). Even though the separation component does not lead to a two-part separation in the mask, the user is still informed of a shared source and is given a general idea of what the region looks like.

UGC 4458

Besides the cases in which the separation component does and does not work, there is also a third possibility: separation is not required. Even in these cases, the module can be applied. UGC 4458 is an example of a cube in which SoFiA has unambiguously assigned all sources to individual objects. When this data cube, mask cube and moment-0 map are plugged into the module, the only system message the user receives is:

```
Galaxies found in your cube: ['NGC 2599', 'KUG 0829+227B']
```

After this the module stops, because neither of these sources share a source ID. This is confirmed by superimposing the contour of the flattened mask cube onto the moment-0 image, shown in figure 6:

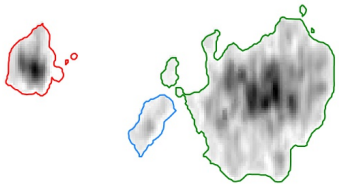

Figure 6 – 2D mask array contour on top of mom-0 map.

While separation might not be necessary for this cube, the module still provides the user with information by alerting them of the contents of the cube.

6 Final Remarks

Discussion

For the development of this module, a sample of 15 cubes from the WHISP survey was selected. These cubes have been chosen because they were at risk of being incorrectly identified by SoFiA. Of these 15, the program identified 7 as correctly separated (i.e. no galaxies share a source ID). The remaining 8 cubes required additional separation.

While in some cases the module leads to a simple n-part division of the mask, where n is the number of galaxies, this is not true for most cases, in which model creation or separation is not possible, or the separation does not lead to desired results. In order to counter this, multiple ways of dilating and determining the base model have been attempted. Having the dilation footprint depend on the average of the *added* pixels, rather than the average of the ellipse plus the added pixels, shows little difference with the original method, although the original method, as explained in section 4, does appear to be somewhat more accurate. While currently implemented to occur only when the watershed model creation has failed, creating a model using the same diameter data used for the footprint is a possibility for all cubes, with the added benefit that even if separation is impossible, the model will provide information about the location and orientation of the galaxies within the mask. In addition to being more accurate, the use of the watershed moment-0 map requires fewer dilation steps and thus less computationally intensive.

Other methods, including scikit-image's built-in watershed function and scikit-learn's spectral clustering function have not proven to be useful in separating the mask accurately.

Despite the sub-optimal separation, the mask returned by the module will provide more detailed information than an ambiguously assigned source and can be used as a basis for further analysis. Furthermore, the results from the identification provide the user with important information about

the contents of the cube and SoFiA's source finding.

One issue that arises in the code is the usage of the exact value of the position angle. By using NED's homogenized data, the coordinates as well as the position angle (PA) are given in the J2000 epoch. While coordinates can easily be converted to another epoch using the Astropy package, the same is not true for the PA, which could lead to the use of a J2000 PA in a cube with a B1950 coordinate system. While the difference in degrees is only minor at low declination, at high declination, the accuracy reduces. Using UGC 1249 and UGC 1256 as an example, according to the coordinate calculator offered on the NED website [5], the difference in PA between J2000 and B1950 epochs is approximately 0.21 degrees (for UGC 1249) and 0.22 degrees (for UGC 1256). For an object at a high declination, e.g. *2MASX J22501987+8958232*, at a RA of 342.707792 degrees and a Dec of 89.973222 degrees, the difference in position angle is much larger: over 16 degrees between J2000 and B1950. For now, the module does not differentiate between J2000 and B1950 position angles and is therefore less effective when used to analyze objects at high declination. While this is not an issue for the ancillary WHISP data, this could potentially be an issue if this module were used in the WALLABY survey, which is set to cover 75% of the sky, at a declination ranging from -90° to 30° [6]. Similarly, with upcoming H I surveys with APERTIF, which are set to have a Declination of more than 30° (Verheijen et al., 2009), the module is capable of separating those sources at lower declination more accurately than those at a higher declination. However, despite the offset of the position angle, the module can still provide important information by identifying observed objects and by providing a basis for separation.

In some cases (for example, the UGC 2941 data cube), there appears to be a small offset (about 2 seconds RA) between the optical center of the galaxy and the center of mass of the H I distribution. This offset is too small to be an effect of the built-in B1950/J2000 converters not working properly, and could be caused by a wrong entry in NED. Furthermore, in some cases (like NGC 3786, one of the galaxies that share a source ID with another), some of the position angles from the diameter data appear to be incorrect. The value from the SDSS isophotal data deviates from the other values, even those from other SDSS analyses. According to the SDSS website [7], the publication of isophotal quantities has stopped since DR8 because of the unreliability of the measurements. In the case of NGC 3786, NED makes use of SDSS DR6, which does contain isophotal quantities. Since the module uses the most recently published data, without looking at the names of the publications, there is a chance the separation component will make use of the SDSS isophotal position angle, which can lead to an offset in the orientation of the model galaxy and the dilation footprint.

The speed at which the program identifies galaxies is dependent on the internet connection of the user and of the state of the NED servers and can therefore vary between users. More important, however, is the object density of the region: in regions that contain many objects, the region query can be significantly slower than in less densely-populated areas. Therefore, some cubes might take longer than others, despite a fast and stable internet connection.

At the moment of writing, separation of the 2D mask is only possible if the galaxies in the model array do not already overlap. Due to the way the dilation in this module is set up, the area in which the model galaxies overlap will not be dilated, leading to an improperly dilated image. Therefore, a check is built in: the module will first shrink the models to half their original size. If separation still is not possible, the user is informed of this via a system message.

In future development, the first obvious improvement is to ensure that separation can not only be done in 2D, but also in 3D. While SoFiA will correctly identify sources that are far apart along the line of sight, if the galaxies are close together, the application could assign a single source to multiple galaxies. Another improvement can be made in the way in which the identification and separation module handles the separation of very small models with a very high average gas number. This leads to wrongly separated 2D masks, as has been illustrated in section 5. By

[5] `https://ned.ipac.caltech.edu/forms/calculator.html`
[6] `http://www.atnf.csiro.au/research/WALLABY/proposal.html`
[7] `http://www.sdss.org/dr12/algorithms/classify/#photo_iso`

tweaking variables or building in systems to catch these cases, it is expected that proper separation is possible.

Conclusion

This article presents the description of a new module for SoFiA (Source Finding Application) that is capable of identification and separation of ambiguously assigned source IDs. By comparing the images generated by SoFiA with information available from NED, it is capable of identifying galaxies within a reduced data cube. The module returns a system message, informing the user of the galaxies that can be found within the cube's spatial and redshift limits. Furthermore, it will recognize source IDs that have been wrongfully assigned to multiple objects and alert the user of these cases. The module will then attempt to separate them by creating a new mask: a 2D version of the original mask, divided between all individual sources. The module generates a 2D model of the region of the sky, using a Gaussian-filtered moment-0 map to which a watershed has been applied, to use as a basis for the dilation process. By having a gas distribution-dependent dilation process, the 2D mask is more accurately divided between the individual galaxies. While the identification provides the user with important information, the separation process is not applicable to all cubes and masks. It can still, however, provide a useful basis for further analysis.

Acknowledgments

I would like to express my gratitude towards Prof. Dr. van der Hulst and Nadine Giese, without whom I could not have successfully developed this module. Furthermore, I would like to acknowledge my friends and family for their encouragement and my fiancé for his unwavering support.
This project has made use of the NASA/IPAC Extragalactic Database (NED) which is operated by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.
I have made use of the WSRT on the Web Archive. The Westerbork Synthesis Radio Telescope is operated by the Netherlands Institute for Radio Astronomy ASTRON, with support of NWO.

References

Duffy, A. R., Moss, A., & Staveley-Smith, L. 2012, PASA, 29, 202

Ginsburg, A., Robitaille, T., Parikh, M., et al. 2013, Astroquery v0.1, http://dx.doi.org/10.6084/m9.figshare.805208.v2, Accessed: 24-05-2016

Hunter, J. D. 2007, Computing In Science & Engineering, 9, 90

Jones, E., Oliphant, T., Peterson, P., et al. 2001–, SciPy: Open source scientific tools for Python, [Online; accessed 2016-06-21]

Serra, P., Westmeier, T., Giese, N., et al. 2015, MNRAS, 448, 1922

The Astropy Collaboration, Robitaille, Thomas P., Tollerud, Erik J., et al. 2013, A&A, 558, A33

van der Hulst, J. M., van Albada, T. S., & Sancisi, R. 2001, in Astronomical Society of the Pacific Conference Series, Vol. 240, Gas and Galaxy Evolution, ed. J. E. Hibbard, M. Rupen, & J. H. van Gorkom, 451

Van Der Walt, S., Colbert, S. C., & Varoquaux, G. 2011, Computing in Science & Engineering, 13, 22

Verheijen, M., Oosterloo, T., Heald, G., & van Cappellen, W. 2009, in Panoramic Radio Astronomy: Wide-field 1-2 GHz Research on Galaxy Evolution, 10

Verheijen, M. A. W., Oosterloo, T. A., van Cappellen, W. A., et al. 2008, in American Institute of Physics Conference Series, Vol. 1035, The Evolution of Galaxies Through the Neutral Hydrogen Window, ed. R. Minchin & E. Momjian, 265–271

7 Appendices

Appendix A: List of cubes

The following cubes have been used in the development of this module. These cubes are part of the WHISP survey and have been downloaded from Westerbork on the Web [8].

Name	Note
UGC 327/2916	Can be found under name: UGC 2916
UGC 1249/1256	Can be found under name: UGC 1256
UGC 1310	
UGC 2942/2941	Can be found under name: UGC 2941
UGC 3407	
UGC 3422/3426	Can be found under name: UGC 3426
UGC 3642	Can be found under name: UGC 3642
UGC 4458	Can be found under name: UGC 4458
UGC 4862	Can be found under name: UGC 4862
UGC 6016/6024	Can be found under name: UGC 6024
UGC 6621/6623	Can be found under name: UGC 6621
UGC 6944	Can be found under name: UGC 6944
UGC 9648	Can be found under name: UGC 9642
UGC 10497/10502	Can be found under name: UGC 10502
UGC 12808	Can be found under name: UGC 12808

[8] http://wow.astron.nl/

Appendix B: Watershed

A watershed is an image processing algorithm, most easily visualized as the flooding of a mountainous landscape, where the mountains represent the amount of neutral hydrogen: the higher the mountain, the more H I there is. As the area is flooded, more and more features are submerged until only 'islands' are left. This process is applied to the moment-0 map, which is first converted to a NumPy array. From this array, low values are set equal to zero, starting with a cap set to $i = 1$. After this, the program counts the number of objects left in the array. If this number is equal to the number of galaxies that share a source ID, the watershed is finished. If not, it raises the cap with 1 to $i = 2$ and repeats this process until the desired number of objects is found. If the desired number of objects cannot be found, the watershed is deemed unsuccessful.

Appendix C: Dilation

Dilation is a morphological operation. When applied to an image (or an array, in this case), a defined kernel (or structuring element) will 'travel' along the edges of the area in which the array is defined (i.e. non-zero values) and add all zero-values within the kernel to the array, by turning the zero-values into non-zero values. This enlarges the original array. The picture below illustrates this operation: the dark blue ellipse is enlarged by an elliptical kernel, adding the outer light blue rim to the image, thus enlarging the original picture. The use of an elliptical kernel with the same orientation as the original is essential to ensure that the enlarged image retains its elliptical shape and proper orientation. A kernel of another shape, e.g. circular, would influence the shape of the original image at a large number of iterations or at a large kernel size, leading to a loss of ellipticity in the dilated image. A kernel with another orientation (i.e. another position angle) would alter the orientation of the dilated image.

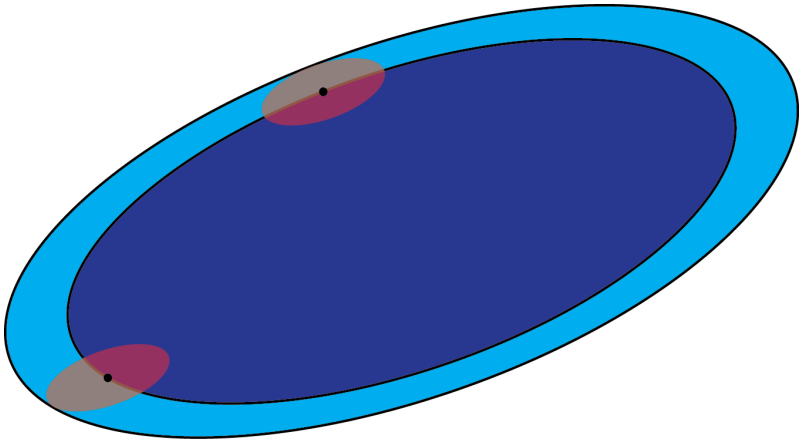

Figure 7 – Dilation through an elliptical kernel

Appendix D: Source code of identification and separation module

```
#!/usr/bin/env python

## EXTERNAL PACKAGES ##
from __future__ import division
import numpy as np
import astropy.units as u
from astropy.wcs import WCS
from astropy.io import fits
from astropy.coordinates import SkyCoord
from astroquery.ned import Ned
from scipy import ndimage
from matplotlib.pyplot import imsave
from timeit import default_timer as timer
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm
import multiprocessing
from kapteyn import maputils


## INPUT ##
mask = fits.open('masks/Run1_masku6944.fits')
maskarray = mask[0].data
file_name = 'cubes/u6944cl.fits'
w = WCS(file_name, naxis=2)
header = fits.getheader(file_name)
temp_mom0 = fits.open('mom0/Run1_mom06944.fits')
mom0 = np.flipud(temp_mom0[0].data)
mask.close()
temp_mom0.close()

## GLOBAL VARIABLES ##
struc = np.ones((3,3))
ellipse = np.empty([header['NAXIS1'], header['NAXIS2']])
info = []

## FUNCTIONS ##

# Searches for objects in NED within a given radius around a
# given position (center of cube)
def nedFind(ra, dec, r):
  c = SkyCoord(ra, dec, unit = 'deg')
  if header['EPOCH'] == 1950:
    sources = Ned.query_region(c, radius=r, equinox='B1950.0')
  else:
    sources = Ned.query_region(c, radius=r)
  return sources


# Finds the radius necessary to cover the entire cube and the coordinates of
# the center pixel using the header info
def findRad():
  xlen = header['NAXIS1']
  ylen = header['NAXIS2']
  midpix1 = header['CRPIX1']
  midpix2 = header['CRPIX2']
  mid = w.all_pix2world(midpix1, midpix2, 1)
  upper_left = w.all_pix2world(0, ylen, 1)
  lower_left = w.all_pix2world(0, 0, 1)
  upper_right = w.all_pix2world(xlen, ylen, 1)
  lower_right = w.all_pix2world(xlen, 0, 1)
  RA_u_l = upper_left[0]
  RA_l_l = lower_left[0]
  RA_u_r = upper_right[0]
  RA_l_r = lower_right[0]
  dec_u_l = upper_left[1]
  dec_l_l = lower_left[1]
  dec_u_r = upper_right[1]
  dec_l_r = lower_right[1]
```

```
69    d1 = np.sqrt(((RA_l_l-RA_u_r)*np.cos(np.radians(dec_l_l)))**2+(dec_l_l-dec_u_r)
        **2)
70    d2 = np.sqrt(((RA_l_r-RA_u_l)*np.cos(np.radians(dec_l_r)))**2+(dec_l_r-dec_u_l)
        **2)
71    d = (d1 + d2)/2
72    r = d/2
73    return mid, r
74
75
76  # Finds redshift (z-axis) limits to filter NED objects by, using header info
77  # from the FITS-file
78  def redshiftFind():
79    c = 2.99792458*(10**8) ## Speed of light in m/s
80    wavelength = 0.2110611405413 ## Wavelength of 21 cm line in m
81    restfreq = c/wavelength
82    fmax = header['CRVAL3'] - (header['CRPIX3'] * header['CDELT3'])
83    fmin = header['CRVAL3'] + ((header['NAXIS3'] - header['CRPIX3']) * header['CDELT3
        '])
84    zmin = (restfreq/fmax) - 1
85    zmax = (restfreq/fmin) - 1
86    return zmin, zmax
87
88
89  # Filters list of NED-found objects using redshifts and type (set to galaxy to
90  # avoid duplicate entries of galaxy groups/pairs/etc and their loose couterparts)
91  def filterZG():
92    temp_gals = []
93    zmin, zmax = redshiftFind()
94    for i in xrange(len(results)):
95      if zmin <= results[i][6] <= zmax and results[i][4] == 'G':
96        newrow = [results[i]]
97        temp_gals.append(newrow)
98    return temp_gals
99
100
101 # Determines pixel location in cube of each NED object by its coordinates
102 def sourcePixFind():
103   nedRA = []
104   neddec = []
105   for i in xrange(len(gals)):
106     nedRA.append(gals[i][0][2])
107     neddec.append(gals[i][0][3])
108     # Check for epoch
109     if header['EPOCH'] == 1950:
110       coords = SkyCoord(nedRA*u.degree, neddec*u.degree, frame='fk5')
111       newcoords = coords.transform_to('fk4')
112     else:
113       newcoords = SkyCoord(nedRA*u.degree, neddec*u.degree, frame='fk5')
114     xcoord, ycoord = w.all_world2pix(newcoords.ra.deg, newcoords.dec.deg, 1)
115   return xcoord, ycoord
116
117
118 # Checks for SoFiA-source ID (non-zero value) in the mask at pixels x,y where
119 # NED found a source, creates list of lists with (name, frame number, ID number)
120 # NB: Indexing as mask[z,y,x]
121 def checkMask():
122   temp_masklist= []
123   temp_filtered_gals = []
124   temp_names_list = []
125   maskdata = mask[0].data
126   x_coord, y_coord = sourcePixFind()
127   for i in xrange(len(gals)):
128     if x_coord[i] <= header['NAXIS1'] and y_coord[i] <= header['NAXIS2']:
129       temp_gals_list = []
130       for j in xrange(maskdata.shape[0]):
131         row=[gals[i][0][1], j, maskdata[j, y_coord[i], x_coord[i]]]
132         temp_gals_list.append(row)
133       temp_masklist.append(temp_gals_list)
134       temp_filtered_gals.append(gals[i])
135       temp_names_list.append(gals[i][0][1])
```

```
136     else:
137       pass
138
139   return temp_masklist, temp_filtered_gals, temp_names_list
140
141
142 # Checks if sublist is not empty and all names are equal within sublist
143 # NB: Input currently should be checkListIntegrity(list[i])
144 def checkListIntegrity(input_list):
145   iterator = []
146   if len(input_list) != 0:
147     for i in xrange(len(input_list)):
148       iterator.append(input_list[i][0])
149     try:
150       iterator = iter(iterator)
151       first = next(iterator)
152       return all(first == rest for rest in iterator)
153     except StopIteration:
154       print iterator, " names within sublist are not all equal."
155   else:
156     print input_list, " your list is empty."
157
158
159 # Prints list of names found in the cube and compare:
160 # Check if first name matches second name. If names do not match, compare
161 # source IDs. If source IDs match, print a warning. Then remove first value
162 # from list and repeat.
163 # Note: compare_list[0][j][2] is first sublist (first NED-found galaxy), j'th
164 # frame number, source ID number at position 2
165 def compareSources(masklist, names, filtered_gals):
166   temp_masklist = list(masklist)
167   temp_names = list(names)
168   temp_filtered_gals = list(filtered_gals)
169   shared_source_gals = []
170   shared_id_list = []
171   print 'Galaxies found in your cube:', temp_names
172   while len(temp_masklist) > 1:
173     for i in xrange(len(temp_masklist)):
174       first_source_list = []
175       second_source_list = []
176       if temp_names[0] != temp_names[i]:
177         for j in xrange(len(temp_masklist[i])):
178           if temp_masklist[0][j][2] != 0.0:
179             first_source_list.append(temp_masklist[0][j][2])
180           if temp_masklist[i][j][2] != 0.0:
181             second_source_list.append(temp_masklist[i][j][2])
182         shared_id=list(set(first_source_list).intersection(second_source_list))
183         if shared_id:
184           shared_id_list.append(shared_id)
185         if shared_id:
186           if temp_filtered_gals[0] in shared_source_gals:
187             if temp_filtered_gals[i] in shared_source_gals:
188               pass
189             else:
190               shared_source_gals.append(temp_filtered_gals[i])
191           else:
192             shared_source_gals.append(temp_filtered_gals[0])
193             shared_source_gals.append(temp_filtered_gals[i])
194           print 'WARNING: SoFiA has given both %s and %s the same sourceID: '%(
    temp_names[0], temp_names[i]), shared_id
195       else:
196         pass
197     else:
198       pass
199     temp_names.pop(0)
200     temp_masklist.pop(0)
201     temp_filtered_gals.pop(0)
202   return shared_source_gals, shared_id_list
203
204
```

```
# Queries NED for diameter data of galaxies that share a source ID in order
# to separate the sources. If no diameter data available, set generic circle
# with radius SQRT(5). (change radius here)
def axisFind():
  diam_data = []
  for i in xrange(len(shared_source_gals)):
    if shared_source_gals[i][0][15]:
      name = shared_source_gals[i][0][1]
      diams = Ned.get_table(name, table="diameters")
      temp_diams = np.array(diams)
      # Sort by most recent published data
      newdiams = sorted(temp_diams,key=lambda l:l[2], reverse=True)
      majaxis = newdiams[0][18]
      axisrat = newdiams[0][20]
      pos_angle = newdiams[0][24]
      # Use second publication if data not available
      if np.isnan(majaxis) or np.isnan(axisrat) or np.isnan(pos_angle):
        majaxis = newdiams[1][18]
        axisrat = newdiams[1][20]
        pos_angle = newdiams[1][24]
      semi_majaxis = majaxis/2
      semi_minaxis = semi_majaxis * axisrat
      row = [name, semi_majaxis, semi_minaxis, 90 + pos_angle]
      diam_data.append(row)
    else:
      print "No diameter data available for: " , shared_source_gals[i][0][1]
      print "Creating a circular kernel for the dilation of: ", shared_source_gals[
    i][0][1]
      radius = 5
      name = shared_source_gals[i][0][1]
      row = [name, radius]
      diam_data.append(row)
  return diam_data


# Uses NED data to find mom0-values at optical centers
def new_ellipseOverlay():
  global ellipse, info
  cent_values = []
  grid = np.zeros((header['NAXIS1'], header['NAXIS2']))
  temp_ellipse = np.copy(grid)
  for i in xrange(len(shared_source_gals)):
    temp_x0 = shared_source_gals[i][0][2]
    temp_y0 = shared_source_gals[i][0][3]

    if header['EPOCH'] == 1950:
      coords = SkyCoord(temp_x0*u.degree, temp_y0*u.degree, frame='fk5')
      coord_transform = coords.transform_to('fk4')
      new_temp_x0 = coord_transform.ra.deg
      new_temp_y0 = coord_transform.dec.deg
      center_coords = w.all_world2pix(new_temp_x0, new_temp_y0, 1)
    else:
      coords = SkyCoord(temp_x0*u.degree, temp_y0*u.degree, frame='fk5')
      new_temp_x0 = coords.ra.deg
      new_temp_y0 = coords.dec.deg
      center_coords = w.all_world2pix(new_temp_x0, new_temp_y0, 1)

    x0 = int(center_coords[0])
    y0 = header['NAXIS1'] - int(center_coords[1])
    center_value = mom0[y0][x0]
    row = y0, x0, center_value, shared_source_gals[i][0][1]
    cent_values.append(row)
  return cent_values


# Creates lists of info required for kernel creation from diameter data
def findInfo():
  info = []
  for i in xrange(len(diam_data)):
    if len(diam_data[i]) == 4:
```

```
274         temp_a = diam_data[i][1] / 3600
275         temp_b = diam_data[i][2] / 3600
276         pos_angle = diam_data[i][3]
277         x0 = cent_values[i][1]
278         y0 = cent_values[i][0]
279         a = temp_a / header['CDELT2']
280         b = temp_b / header['CDELT2']
281         row = [(i+1), diam_data[i][0], x0, y0, a, b, pos_angle]
282         info.append(row)
283       elif len(diam_data[i]) == 2:
284         radius = diam_data[i][1]
285         row = [(i+1), diam_data[i][0], radius]
286         info.append(row)
287     return info
288 
289 
290 def ellipseOverlay(factor):
291     x, y = np.mgrid[:header['NAXIS1'], :header['NAXIS2']]
292     grid = np.zeros((header['NAXIS1'], header['NAXIS2']))
293     info = []
294     for i in xrange(len(diam_data)):
295       if len(diam_data[i]) == 4:
296         temp_x0 = shared_source_gals[i][0][2]
297         temp_y0 = shared_source_gals[i][0][3]
298         temp_a = diam_data[i][1] / 3600
299         temp_b = diam_data[i][2] / 3600
300         pos_angle = diam_data[i][3]
301 
302         if header['EPOCH'] == 1950:
303           coords = SkyCoord(temp_x0*u.degree, temp_y0*u.degree, frame='fk5')
304           coord_transform = coords.transform_to('fk4')
305           new_temp_x0 = coord_transform.ra.deg
306           new_temp_y0 = coord_transform.dec.deg
307           center_coords = w.all_world2pix(new_temp_x0, new_temp_y0, 1)
308         else:
309           coords = SkyCoord(temp_x0*u.degree, temp_y0*u.degree, frame='fk5')
310           new_temp_x0 = coords.ra.deg
311           new_temp_y0 = coords.dec.deg
312           center_coords = w.all_world2pix(new_temp_x0, new_temp_y0, 1)
313 
314         x0 = center_coords[0]
315         y0 = center_coords[1]
316         a = factor*(temp_a / header['CDELT2'])
317         b = factor*(temp_b / header['CDELT2'])
318         row = [(i+1), diam_data[i][0], x0, y0, a, b, pos_angle]
319         info.append(row)
320         ellipse = (((x-x0)*np.cos(np.radians(pos_angle)) + (y-y0)*np.sin(np.radians(
        pos_angle)))**2 / (a**2)) + (((x-x0)*np.sin(np.radians(pos_angle)) - (y-y0)*np.
        cos(np.radians(pos_angle)))**2 / (b**2)) <= 1
321         if i == 0:
322           newellipse = (i+1) * ellipse
323           overlay = np.add(grid, newellipse)
324         else:
325           newellipse = (i+1) * ellipse
326           overlay = np.add(overlay, newellipse)
327       elif len(diam_data[i]) == 2:
328         temp_x0 = shared_source_gals[i][0][2]
329         temp_y0 = shared_source_gals[i][0][3]
330         radius = factor*(diam_data[i][1])
331 
332         if header['EPOCH'] == 1950:
333           coords = SkyCoord(temp_x0*u.degree, temp_y0*u.degree, frame='fk5')
334           coord_transform = coords.transform_to('fk4')
335           new_temp_x0 = coord_transform.ra.deg
336           new_temp_y0 = coord_transform.dec.deg
337           center_coords = w.all_world2pix(new_temp_x0, new_temp_y0, 1)
338         else:
339           coords = SkyCoord(temp_x0*u.degree, temp_y0*u.degree, frame='fk5')
340           new_temp_x0 = coords.ra.deg
341           new_temp_y0 = coords.dec.deg
```

```
342         center_coords = w.all_world2pix(new_temp_x0, new_temp_y0, 1)
343
344       x0 = center_coords[0]
345       y0 = center_coords[1]
346       circle = (x-x0) ** 2 + (y-y0) ** 2 < radius
347       if i == 0:
348         newellipse = (i+1) * circle
349         overlay = np.add(grid, newellipse)
350       else:
351         newellipse = (i+1) * circle
352         overlay = np.add(overlay, newellipse)
353   overlay = np.rot90(overlay)
354   return overlay, info
355
356
357 # Excludes sources that are not shared, flatten mask to 2D and give uniform value
358 def flattenMask():
359   shared_id_list = compare_sources[1]
360   #maskarray = mask[0].data
361   for i in xrange(len(shared_id_list)):
362     temp_flat_mask = np.where(maskarray == shared_id_list[i], 1, 0)
363     sum_flat_mask = temp_flat_mask.sum(axis=0)
364     flat_mask = np.where(sum_flat_mask != 0, 1, 0)
365     flat_mask = np.flipud(flat_mask)
366     return flat_mask
367
368
369 # Finds minimum required size for dilation (optimization step)
370 def subarrayShape():
371   flat_mask = flattenMask()
372   locs = ndimage.find_objects(flat_mask)
373   return locs
374
375
376 # Finds the kernel required for dilation of the ellipses, if no kernel available,
377 # define generic circle with r = SQRT(5) (set on line 251)
378 def findKernel(factor, i):
379   if len(info[i]) == 7:
380     all_zero = False
381     a = factor*info[i][4]
382     b = factor*info[i][5]
383     pos_angle = 90 + info[i][6]
384     dil = 1
385     dd  = 2*dil+1
386     x, y = np.indices((dd,dd))-dil
387     footprint = (((x*np.cos(np.radians(pos_angle)) + y*np.sin(np.radians(pos_angle)
    ))**2 / (a**2)) + ((x*np.sin(np.radians(pos_angle)) - y*np.cos(np.radians(
    pos_angle)) )**2 / (b**2)) <= 1).astype(int)
388     top = footprint[0]
389     bot = footprint[dd-1]
390     left = footprint[:,0]
391     right = footprint[:,dd-1]
392     while all_zero == False:
393       if np.sum(top) == 0 and np.sum(bot) == 0 and np.sum(left) == 0 and np.sum(
    right) == 0:
394         all_zero = True
395         dil = dil-1
396         new_dd  = 2*dil+1
397         new_x, new_y = np.indices((new_dd,new_dd))-dil
398         new_footprint = (((new_x*np.cos(np.radians(pos_angle)) + new_y*np.sin(np.
    radians(pos_angle)))**2 / (a**2)) + ((new_x*np.sin(np.radians(pos_angle)) -
    new_y*np.cos(np.radians(pos_angle)) )**2 / (b**2)) <= 1).astype(int)
399         return new_footprint
400       else:
401         dil = dil+1
402         new_dd  = 2*dil+1
403         new_x, new_y = np.indices((new_dd,new_dd))-dil
404         new_footprint = (((new_x*np.cos(np.radians(pos_angle)) + new_y*np.sin(np.
    radians(pos_angle)))**2 / (a**2)) + ((new_x*np.sin(np.radians(pos_angle)) -
    new_y*np.cos(np.radians(pos_angle)) )**2 / (b**2)) <= 1).astype(int)
```

```
405         top = new_footprint[0]
406         bot = new_footprint[new_dd-1]
407         left = new_footprint[:,0]
408         right = new_footprint[:,new_dd-1]
409   elif len(info[i]) == 3:
410     dil = 3
411     dd  = 2*dil+1
412     x, y = np.indices((dd,dd))-dil
413     circle = x ** 2 + y ** 2
414     footprint = circle < info[i][2]
415     return footprint
416
417
418 # Dilates each object individually using a kernel with a size dependant
419 # on the average of the amount of gas in the mom0 image
420 def new_dilate(grid_input):
421   dilated_grid = np.copy(grid_input)
422   loc = subarrayShape()[0]
423   flatmask = flattenMask()
424   prev_avg = 1
425   fully_dilated = False
426   while fully_dilated == False:
427     print 'dilating...'
428     temp_dilated_grid = np.copy(dilated_grid)
429     for i in xrange(len(shared_source_gals)):
430       ellipse = np.where(dilated_grid == i+1, 1, 0)
431       cut_out = np.multiply(ellipse, mom0)
432       nonzero = np.nonzero(cut_out)
433       avg = np.absolute(cut_out[nonzero].mean())
434
435       if avg <= prev_avg:
436         footprint = findKernel(1.5, i)
437         print i, avg
438       elif np.isnan(avg):
439         footprint = findKernel(1.5, i)
440         print i, 'avg = nan'
441       else:
442         footprint = findKernel(1*avg, i)
443         print i, avg
444       prev_avg = avg
445
446       dilated_elipse_grid = ndimage.morphology.binary_dilation(ellipse, structure =
      footprint)
447       dilated_elipse_grid = (i+1)*dilated_elipse_grid
448       for x_coord in xrange(len(ellipse)):
449         for y_coord in xrange(len(ellipse)):
450           if temp_dilated_grid[y_coord][x_coord] == 0:
451             temp_dilated_grid[y_coord][x_coord] = dilated_elipse_grid[y_coord][
    x_coord]
452       temp_dilated_grid = np.multiply(temp_dilated_grid, flatmask)
453
454     if np.array_equal(dilated_grid, temp_dilated_grid):
455       fully_dilated = True
456     else:
457       dilated_grid = temp_dilated_grid
458   print 'done dilating!'
459   return dilated_grid
460
461
462 # Checks for matches with watershed-mom0 image at the optical center of the
      galaxies
463 def check_values(image):
464   values = []
465   temp_model = np.copy(image)
466   for i in xrange(len(cent_values)):
467     coord = cent_values[i][0], cent_values[i][1]
468     value = temp_model[coord]
469     values.append(value)
470   if any([v == 0 for v in values]):
471     return False
```

```
472   else:
473     return True
474
475
476 # Watersheds mom0 image. Prints sys-msg if watershedding impossible (i.e. if the
477 # watershed creates less objects than shared-source galaxies)
478 def watershed_model(sigma):
479   flat_mask = flattenMask()
480   image = mom0.byteswap().newbyteorder()
481   temp_image = np.multiply(image, flat_mask)
482   array, num = ndimage.measurements.label(temp_image, structure=struc)
483   temp_image = ndimage.gaussian_filter(temp_image, sigma=sigma)
484   new_image = ndimage.gaussian_filter(temp_image, sigma=sigma)
485   check = check_values(temp_image)
486   i = 1
487   while num != len(shared_source_gals):
488     new_image = np.where(temp_image >= i, 1, 0)
489     array, num = ndimage.measurements.label(new_image, structure=struc)
490     if num < len(shared_source_gals):
491       break
492     i += 1
493     check = check_values(new_image)
494   return new_image
495
496
497 # Creates model by watershedding Gauss-filtered mom0 image. Uses variable sigma
498 # if optical center does no coincide with model before sigma=15, stops
499 def create_model():
500   new_image = watershed_model(1)
501   check = check_values(new_image)
502   if check == True:
503     return new_image
504   else:
505     sigma = 1
506     while check == False:
507       new_image = watershed_model(sigma)
508       check = check_values(new_image)
509       sigma += 0.1
510       if sigma >= 15:
511         print 'watershedding unsuccessful (no match /w optical center at sigma=15)'
512         break
513     return new_image
514
515
516 # Assigns correct value to each individual object, necessary for dilation.
517 def assign_values(input_model):
518   marker_list = []
519   for i in xrange(len(shared_source_gals)):
520     marker_list.append(i+11)
521   model = np.copy(input_model)
522   array, num = ndimage.measurements.label(model, structure=struc)
523   locs = ndimage.find_objects(array)
524   for i in xrange(len(locs)):
525     for j in xrange(len(marker_list)):
526       obj = model[locs[i]]
527       if marker_list[j] in obj:
528         value = info[j][0]
529         model[locs[i]] = np.where(obj, value, 0)
530   return model
531
532
533 # Assigns markers to objects for correct identification of objects, based on
534 # distance from optical center (from NED) to center of mass of HI object.
535 def assign_neighbor_marker(input_obj):
536   obj = np.copy(input_obj)
537   array, num = ndimage.measurements.label(obj, structure=struc)
538   locs = ndimage.find_objects(array)
539   CoMs = []
540   coords = []
541   for i in xrange(num):
```

```
542     temp_obj = np.where(array == (i+1), 1, 0)
543     center_of_mass = ndimage.measurements.center_of_mass(temp_obj)
544     CoMs.append(center_of_mass)
545     # Find distance to optical center
546   CoMs_array = np.asarray(CoMs)
547   for i in xrange(len(shared_source_gals)):
548     coord = [cent_values[i][0], cent_values[i][1]]
549     row = coord
550     coords.append(row)
551   coords_array = np.asarray(coords)
552   for i in xrange(len(CoMs)):
553     distances = []
554     for j in xrange(len(coords)):
555       dist = np.linalg.norm(CoMs_array[i] - coords_array[j])
556       distances.append(dist)
557     dist_array = np.array(distances)
558     min_dist_index = dist_array.argmin()
559     obj[CoMs[i]] = (min_dist_index+11)
560   return obj
561
562
563 # Checks what type of model creation is required; if objects cannot be
564 # identified after a Gaussian filter/watershed, revert to NED model
565 def check_ModelCreation():
566   marker_list = []
567   temp_model = create_model()
568   model = assign_neighbor_marker(temp_model)
569   for i in xrange(len(shared_source_gals)):
570     marker_list.append(i+11)
571   if all(markers in model for markers in marker_list):
572     print 'Watershed successful!'
573     model = assign_values(model)
574     return model
575   else:
576     print 'Watershed unsuccessful, using NED model'
577     model = ellipseOverlay(1)[0]
578     return model
579
580
581 # Checks if separation is necessary. If necessary, checks if separation is
582 # possible by comparing number of objects in model to number of galaxies
583 # in list. (models that overlap cannot be separated)
584 def checkSeparation():
585   model = check_ModelCreation()
586   flat_mask = flattenMask()
587
588   ##Checks##
589   imsave('ws_6944_plak.png', np.add(flat_mask, model))
590   imsave('ws_6944_modelL.png', model)
591   imsave('ws_6944_mom0.png', mom0)
592   imsave('ws_6944_mask.png', flat_mask)
593   if shared_source_gals:
594     array, num = ndimage.measurements.label(model, structure=struc)
595
596     if len(shared_source_gals) == num:
597       print 'Separation necessary and possible.'
598       print 'Starting separation...'
599       start2 = timer()
600       dilation = new_dilate(model)
601       imsave('ws_6944_dil.png', dilation)
602       overlay = np.multiply(flat_mask, dilation)
603       imsave('ws_6944sep.png', overlay)
604       stop2 = timer()
605       print 'Separation: ', (stop2-start2), 's'
606       return overlay
607     else:
608       print 'Model creation failed, attempting smaller model..'
609       new_model = ellipseOverlay(0.5)[0]
610       imsave('ws_6944_plakS.png', np.add(flat_mask, new_model))
611       imsave('ws_6944_modelS.png', new_model)
```

```
612       new_array, new_num = ndimage.measurements.label(new_model, structure=struc)
613       if len(shared_source_gals) == new_num:
614         print 'Separation necessary and possible.'
615         print 'Starting separation...'
616         start2 = timer()
617         dilation = new_dilate(new_model)
618         imsave('ws_6944_dil.png', dilation)
619         overlay = np.multiply(flat_mask, dilation)
620         imsave('ws_6944sep.png', overlay)
621         stop2 = timer()
622         print 'Separation: ', (stop2-start2), 's'
623         return overlay
624       else:
625         print 'WARNING: separation not possible.'
626   else:
627     print 'Separation not required!'
628
629
630 # ------------------------------------------------------------------------ #
631 ## Run code ##
632 ## IDENTIFICATION
633 start1 = timer()
634 # Generate NED data table containing all objects in cube
635 r = findRad()[1]
636 midCoord = findRad()[0]
637 result_table = nedFind(midCoord[0], midCoord[1], r*u.degree)
638 results = np.array(result_table)
639 # Filter results
640 gals = []
641 gals = filterZG()
642 # Compare to mask
643 masklist = []
644 filtered_gals = []
645 names_list = []
646 masklist, filtered_gals, names_list = checkMask()
647 # Check list integrity
648 for i in xrange(len(masklist)):
649   checkListIntegrity(masklist[i])
650 # Compare found source IDs
651 compare_sources = compareSources(masklist, names_list, filtered_gals)
652 shared_source_gals = compare_sources[0]
653 # Identification finished
654 stop1 = timer()
655 print 'Identification: ', (stop1-start1), 's'
656
657 ## SEPARATION
658 # NED query for diameter data
659 cent_values = new_ellipseOverlay()
660 diam_data = axisFind()
661 # Generate information (axis length, pos_angle)
662 info = findInfo()
663 # Attempt separation
664 checkSeparation()
```