VLASS Project Memo #15

VLASS Imaging Project: Proposed Workflow

John J. Tobin, Josh Marvil, Juergen Ott, and Sanjay Bhatnagar (Project Director)

May 18, 2020

1 Introduction

The VLASS Imaging Project (VIP) has developed a new workflow for the creation of Single Epoch (SE) continuum images from the Very Large Array Sky Survey (VLASS). We have implemented this workflow outside the pipeline framework as CASA scripts using standalone calls to `tclean`, other CASA tasks, and external scripts that were required to support the new workflow. As such, this workflow must now be implemented into the pipeline and VLASS Workflow System to support SE Continuum processing in a production environment. This workflow is appropriate for VLASS images that contain sources with peak intensities up to ~500 mJy beam^{-1}, additional heurstic development is required for images containing brighter sources.

The major departure of the VIP Imaging workflow is that VLASS will no longer make use of the `automasking` algorithm in `tclean` and will instead use static masks that are generated from the VLASS Epoch 1 Quicklook component catalog and a mask generated as part of the workflow. The additional mask is created using the `pyBDSF` package (source-finding software widely used in radio astronomy community), to mask sources missed in the conservatively produced Quicklook component catalog.

The script which performs the recommended workflow is available from:
`https://gitlab.nrao.edu/jmarvil/vlass-imaging-project/-/blob/master/vip_script.py`

The script is also in Section 5 of this document, annotated with comments indicating the code associated with the steps outlined in Section 3.

2 Summary of Major Recommendations

2.1 Masking

The imaging workflow will no longer use the auto-masking algorithm as implemented in `tclean`. This was found to be a culprit for the frequent divergence of images in the testing of the original imaging pipeline.

Masking of sources is still essential to the proper deconvolution of images, and we will use two tiers of masking. A Tier-1 mask is generated from the component catalog generated from the VLASS Quicklook images. This component catalog is conservatively generated, avoiding the inclusion of significant artifacts and does not include QA rejected images. The Tier-1 mask will be used for model creation, enabling proper setting of weights, RFI flagging, and self-calibration. The Tier-2 mask is then generated after self-calibration, from the image produced after an initial cleaning of the self-calibrated data using the Tier-1 mask. This enables the Tier-2 mask to include fainter sources and extended structure missing from the Tier-1 mask. The creation of the Tier-2 mask requires the use of `pyBDSF` to identify additional components not included in the Tier-1 mask and create the final mask, a linear combination of the Tier-1 and Tier-2 masks that is used to complete the imaging process.

2.2 Point Spread Functions

The `tclean` task in CASA uses the AW-Projection algorithm to compute the wide-band PSFs at the pointing center, where the PB has no frequency dependence. The wide-band PSF computed at this location is therefore appropriate for single-pointing wide-band imaging. Mosaic imaging spreads the PB's frequency dependence to the entire mosaic region and renders the PSF as computed by default in `tclean` inaccurate for wide-band continuum mosaic imaging (the PB frequency dependence gets embedded in the PSF, which in-turn leads to wide-spread error in the wide-band modeling of the sky emission).

The AW-Projection implementation in CASA allows switching-off of the frequency-dependent PBs and we first compute the PSF using the parameter `wbawp=False`. Then in subsequent imaging steps, we replace the PSF generated for the images (which must use `wbawp=True` for imaging) with the PSF created using `wbawp=False`. However, with the current implementation in CASA the generation of the PSF with `wbawp=False` and the images with `wbawp=True` requires different Convolution Function Caches (CFCaches).

Note that it is preferable for CASA to handle these PSF issues internally. This would eliminate the need for two CFCaches and the PSF replacement workaround, but development priorities will prevent these issues from being solved until after CASA 6.2. Thus, the pipeline must work around the CASA limitation for the the implementation of this workflow.

2.3 Convolution Function Caches

CFCache are an integral part of the deconvolution process using the `gridder='awproject'`. These cannot currently be accessed efficiently using the Lustre file system and are accessed from `/mnt/scratch/cfcache` on the VLASS cluster nodes. As implemented in CASA 5.6.3, two CFCaches are needed one for the creation of the PSF with `wbawp=False` and another CFCache created with `wbawp=True` for the deconvolution of the images.

2.4 Image and Pixel Size

The image sizes are recommended to be 16384x16384 and the pixel size is still recommended to be $0''.6$ per pixels. This larger image size is in part driven by our recommendation to lower the `pblimit` parameter from 0.2 to 0.02 to reduce the image truncation at low primary beam levels and the larger images avoid the effects of aliasing.

While we recommend these larger images at present, the pixel size and image sizes are still being investigated to determine if smaller pixels (and hence larger images) are needed or if the image size could be modestly reduced. The issue being that PSF oversampling with a $0\farcs6$ cell size is on the border of what is typically considered acceptable, especially at the highest frequencies of S-band (4 GHz). If we require 3 pixels to sample the main lobe of the PSF, considering the central wavelength of S-band, the PSF is sampled with 3 pixels using $0\farcs6$ pixels, but at the high frequency end of the band, $0\farcs43$ pixels are needed to sample the PSF with 3 pixels. The choice here is important because it will also dictate the image size and PSF sampling for the coarse cubes.

2.5 Self-Calibration and Statistical Weights

Use of self-calibration was planned for the VLASS SE imaging pipeline previously, but as part of VIP we confirm that it is a necessity for producing the highest quality images, even in low-dynamic range fields. The method of calculating statistical weights is recommended to be changed from the current method (time dependent and per spectral window) to channelized weights (time independent and per channel). The new heuristics are helpful for obtaining accurate flux densities toward bright sources.

2.6 Configurable Parameters

While the VIP team has made strides to determine parameters that will work for a large number of images, it is very likely that there will be images that require parameter adjustments to enable successful imaging. As such, we have identified a number of parameters that must be exposed to the pipeline for manipulation by the Data Analysts for running the imaging workflows. Furthermore, further optimization of some parameters is still being explored and the recommended values may change as further parameter studies are conducted.

Imaging parameters:

- `robust`: default=1 or 1.5, changed at different points in the workflow
- `imsize`: default=16384
- `cfcache`: CFCache location
- `cell`: pixel size default=0.6″
- `uvtaper`: default=”
- `uvrange`: default=’<12km’
- `nsigma`: default=3.0
- `niter`: default=20000
- `scales`: default=[0,5,12]
- `cycleniter`: default=5000
- `cyclefactor`: default=3
- `pblimit`: default=0.02

Self-calibration:

- Task `gaincal`, parameter `combine`: default=`’spw,field’`

- Task `gaincal`, parameter `spw`: default=`''`
- Task `gaincal`, parameter `solint`: default = `'inf'`
- Task `gaincal`, parameter `minsnr`: default = `'5.0'`
- Task `applycal`, parameter `interp`: default= `'nearest'`
- Task `applycal`, parameter `spwmap`: default=`[2]*18`

Statistical Weighting (task `statwt`):

- Parameter `combine`: default=`'field,scan,state,corr'`
- Parameter `chanbin`: default=`1`
- Parameter `timebin`: default=`'1yr'`
- Parameter `datacolumn`: default=`'residual_data'`

Tier-2 Mask Creation:

- Input image: default=`'image'` alternate `'residual'`
- Do artifact rejection: default=`'True'`

3 Proposed Workflow

The VIP has developed a recipe for the flow of the pipeline incorporating the changes recommended for the successful deconvolution of VLASS data. A high-level outline of the recipe is provided below, and further details of each step are found in subsections below. The necessary steps resulting from the VIP study are 3 through 14, and the other steps are straightforward operations that the pipeline already performs.

1. Import data (standard pipeline)
2. Split data from full VLASS MS, reset weights (standard pipeline)
3. Create uniform and robust PSFs using wbawp=False (new procedure)
4. Make Tier-1 mask from QL catalog (new procedure)
5. Make uniform weight image with $3''$ taper, fill model column (new procedure)
6. RFI flagging using vlass-imaging mode (standard pipeline)
7. Run Statwt on residual data (recommend task modifications)
8. Self-calibration (recommend task modifications)
9. Create new uniform and robust PSFs using wbawp=False (new procedure)
10. Make new uniform weight image, tapered to $3''$ and smooth to $5''$ (new procedure)
11. Create Tier-2 mask from smoothed image using pyBDSF (new procedure)
12. Combine Tier-1 mask with Tier-2 mask (new procedure)

13. Make robust weight image, cleaned using Tier-1 mask (new procedure)
14. Resume robust image using combined mask, finishing deconvolution (new procedure)
15. PB Correct images (standard pipeline)
16. Make RMS images (standard pipeline)
17. Cutout images (standard pipeline)
18. Analyze Alpha (standard pipeline)
19. Export Products (standard pipeline)

3.1 Step 3: Create Uniform and Robust PSFs using `wbawp=False`

As discussed in section 2.1, CASA 5.6.3 by default creates a PSF for an `awproject` mosaic image that is not optimal. Thus, PSFs need to be created with the `tclean` parameter `wbawp` set to `False`. These PSFs must then be used to replace the PSFs that are generated for the images themselves. These PSFs require separate CFCaches for their creation. The image PSFs are replaced by first initializing an image with no cleaning (analogous to iter0 in pipeline imaging), deleting the `*.psf.tt*` files associated with the image, and replacing them with the `wbawp=False` PSFs. Then, `tclean` can be restarted and deconvolution begun.

3.2 Step 4: Create Tier-1 Mask from Quicklook Component Catalog

The VLASS Quicklook component catalog (`/home/vlass/packages/VLASS1Q.fits`) is used to generate a first mask for the cleaning required for steps 5, 10 and 13. The code used by VIP to generate the mask is located at `/home/vlass/packages/vip/mask-from-catalog/mask_from_catalog.py`. This function is imported into CASA and run within the CASA environment, producing a mask file that can be used in imaging. A typical call is:

```
mask_from_catalog(inext='iter0.psf.tt0', outext="QLcatmask.mask",
catalog_search_size=2.0).
```

The parameter `inext` is simply an input CASA image the spans the full field of view where masks should be drawn; the input image is only used to obtain the image center, `outext` is the name of the output mask, and `catalog_search_size` is the circle (or is it a box) size to search for catalog components for masking.

3.3 Step 5: Create Uniform Weight image with Tapering and Fill Model Column

An initial sky model is required in order to perform steps 6, 7, and 8 and an initial image is created from the data to generate this model. This image is created using uniform weighting, but tapered to 3″. The use of uniform weighting is recommended here because the positive sidelobe levels are reduced with uniform weighting with respect to `robust=1.0` weighting; tapering ensures that we do not undersample the resulting PSF of the image.

Cleaning should go down to 20σ or 1% of the image peak flux to avoid cleaning deconvolution artifacts around bright sources, but also clean sufficient flux per row to create a good enough sky model for RFI flagging, computation of weights, and self-calibration.

Once deconvolution is complete, the model column of the measurement set needs to be filled. Writing to the model column must be done serially; however, CASA 6.1 internally solve the problems associated with `tclean` writing the model column in parallel mode. Currently in CASA 5.6.3, writing the model is slow in serial because `tclean` still must perform many computations, in addition to the act of writing to disk.

3.4 Step 6: Run RFI Flagging

With the model column written, RFI flagging can take place on the residual data. The pipeline task `hifv_checkflag` (`checkflagmode='vlass-imaging'`) incorporates the desired heuristics already and fits within the recommended workflow. An alternative RFI flagging heuristic was used in VIP to avoid dependence on the pipeline; we are not recommending adoption of these heuristics.

3.5 Step 7: Run `statwt`

Statistical weights need to be properly calculated for the VLASS data, which requires a sky model in order to properly weight the data and not down weight bright sources. Thus, VIP still desires to run `statwt` on the `residual_data` (data column minus model). The current implementation of `statwt` (time dependent and per spw) in the pipeline may be insufficient for the needs of VLASS as demonstrated in imaging tests. Instead, running `statwt` in channelized mode (time independent) can yield superior results and enable VLASS imaging to have better flux density accuracy on bright sources. Our recommended call to `statwt` is:

```
statwt(vis=vis, combine='field,scan,state,corr', chanbin=1,timebin='1yr',
datacolumn='residual_data').
```

It is preferable provide the ability to run `statwt` in both the way it has been run and in channelized mode with a parameter setting.

3.6 Step 8: Run Self-Calibration

Self-calibration of VLASS data is essential for correcting some imaging artifacts that are present in the data. We recommend only small changes to the self-calibration task currently defined in the pipeline. An example tasks `gaincal` and `applycal` call are:

```
gaincal(vis=vis, caltable='g.0',gaintype='G', calmode='p',
refant=refantlist, combine='field,spw', minsnr=5)
```

```
applycal(vis=vis, calwt=False, applymode='calonly', gaintable='g.0', spwmap=18*[2],
interp='nearest').
```

Following are the main changes we recommend:

1. Increase the value of the `minsnr` parameter from 1 to 5. This will ensure that poor solutions are not applied to the data. We will use the `applycal` option `applymode='calonly'` which will not flag data for which the self-calibration solutions are flagged and will leave the data as-is. The solutions with the setting of `combine='field,spw'` will create solutions per row in the VLASS image and `gaintype='G'` will create a solution per polarization.

2. Change the interpolation mode for task `applycal` to simply `nearest` instead of `nearestPD`. The `'PD'` mode appended to the `interp` argument for the task `applycal` is not recommended here because the phase solutions are scaled with frequency. While `'PD'` is appropriate for tropospheric phase distortions that increase as $\sim \nu^1$, it is not appropriate the ionosphere, which is likely a main contributor to phase variations in S-band, where the variations increase as ν^{-1}, nor is it appropriate for electronic phase changes which we presume do not scale with frequency. `gaintype='G'` seems most appropriate for S-band because phase variations are most likely to be due to the ionosphere and can be polarization dependent and some polarization-dependent primary beam uncertainties are absorbed into the self-calibration solutions.

3.7 Step 9: Create new uniform and robust PSFs with `wbawp=False`

After running the task `statwt`, RFI flagging, and self-calibration, new PSFs must be created with `wbawp=False`. This because the additional flagging and re-weighting of the data are expected to alter the PSF, necessitating the creation of new PSFs to use in the final deconvolution. A new PSF for the uniform weighted image tapered to $3''$ is required, as is a new PSF for the robust value to be used in the final imaging (`robust = 1`). Note that `robust=1.5` was originally planned for VLASS SE imaging, however, early-on VIP adopted `robust=1`, but we still expect this workflow to work with `robust=1.5`.

3.8 Step 10: Make new uniform weight image

A new uniform weighted image, tapered to $3''$ must be generated from the corrected data, cleaning down to 3σ. This image is required to create the Tier-2 mask from self-calibrated data. The self-calibrated data will have the highest degree of image artifact reduction and the uniform weighting will produce the lowest positive sidelobes.

3.9 Step 11: Create Tier-2 mask using `pyBSDF`

The creation of the Tier-2 mask uses the uniform weight image created in Step 10 and smooths it with a $5''$ Gaussian. The smoothed image will enable better sidelobe rejection and aid in the detection of low-surface brightness features in the mask. We recommend that the Tier-2 mask be derived from the restored image, but we also recommend that an option be present to create the the mask from the residual image if desired.

To create the Tier-2 mask, the program pyBDSF (`https://www.astron.nl/citt/pybdsf/`) is run on the smoothed image from the call:
`subprocess.call(['run_bdsf.py', imagename_base+'iter1b.image.smooth5.fits'], env='PYTHONPATH':'')`.

This runs an external instance of `pyBDSF` outside the CASA environment to generate the Tier-2 mask. We are currently running `pyBDSF` on Python 2.7, but the package is available compatible with Python 3.

The output from `pyBDSF` does include some false positives that need to be rejected whenever possible. To do this, we use a script `edit_pybdsf_islands.py` which is called via:
`edit_pybdsf_islands(catalog_fits_file=imagename_base+'iter1b.image.smooth5.cat.fits')`.

The filtered mask is then turned into a CASA mask image using the same program that generated the Tier-1 mask via:

```
    mask_from_catalog(inext=inext,outext="secondmask.mask",
catalog_fits_file=imagename_base+'iter1b.image.smooth5.cat.edited.fits',
catalog_search_size=2.0).
```

The parameter `inext` is the PSF image, required to match the mask's coordinate system to that of the images it will be applied to, `outext` is the desired name of the output mask, `catalog_fits_file` is the table of mask positions with filtering applied to reject likely artifacts, and `catalog_search_size` is the radius in degrees to search for matches. This is set to 2.0° because the images can be up to 2°×2°.

3.10 Step 12: Combine Tier-1 and Tier-2 Masks

We then take the union of the Tier-1 and Tier-2 masks in order to recover any sources that may have been filtered out or missed in the Tier-2 mask generation process. This combined mask represents the final mask that will be used for cleaning the VLASS SE cont images.

3.11 Step 13: Create Image with `robust=1` weighting using Tier-1 Mask

Next, the generation of the actual image product begins with the creation of the `robust=1` image. The PSF used in this image needs to be replaced with the `robust=1` PSF generated using `wbawp=False` in Step 9. The cleaning of this image is started with only the Tier-1 mask. This is because not all artifacts are rejected from the Tier-2 mask and starting with the Tier-1+Tier-2 mask from the start can lead to imaging problems. This image is cleaned down to 3σ.

3.12 Step 14: Resume Cleaning Robust=1 Image with Tier-1+Tier-2 Mask

Once cleaning has completed on the `robust=1` image using the Tier-1 mask, cleaning is resumed using the Tier1+Tier2 mask. The image is then cleaned down to 3σ, representing what will become the final image product.

3.13 Steps 15-19: Final Product Creation

Following Step 14, the final processing is performed on the images to create the products. These steps already exist in the pipeline and their use was not examined by VIP. The images are primary beam corrected (Step 15), the RMS image is generated (Step 16), the $1^\circ\times1^\circ$ image is cut out of the full $2^\circ\times2^\circ$ image (Step 17), the spectral index of the brightest source is recorded (Step 18), and the $1^\circ\times1^\circ$ image and RMS image are exported as FITS files (Step 19).

4 Summary

The VIP recommended workflow will require changes to the previously planned SE imaging recipe and the VIP team is available for consultation in the development process. Steps are added in comments that refer to the VIP workflow steps outlined in Section 3.

5 Script

The script which performs the recommended workflow is available from:
https://gitlab.nrao.edu/jmarvil/vlass-imaging-project/-/blob/master/vip_script.py

```
vis                =  'VLASS1.2.sb36484946.eb36542800.58574.4235612037_ptgfix_split.ms'
phasecenter        =  'J2000 13:32:4.836 +16.30.0.0000'
field              =  ''
spw                =  ''
antenna            =  ''
scan               =  ''

imagename_base     =  'VIP_'
imsize             =  16384
cell               =  '0.6arcsec'
reffreq            =  '3.0GHz'
uvrange            =  '<12km'
intent             =  'OBSERVE_TARGET#UNSPECIFIED'
wprojplanes        =  32
usepointing        =  True

cfcache='/mnt/scratch/cfcache/cfcache_spw2-17_imsize16384_cell0.6arcsec_w32_conjT.cf'
cfcache_nowb='/mnt/scratch/cfcache/cfcache_spw2-17_imsize16384_cell0.6arcsec_w32_conjT_psf_wbawp_False.cf'

inext='iter0.psf.tt0'

import os
import re
import shutil
import subprocess

execfile('/home/vlass/packages/vip/mask-from-catalog/mask_from_catalog.py')
execfile('/users/jmarvil/scripts/edit_pybdsf_islands.py')



def run_tclean( image_iter, datacolumn='data', cfcache=cfcache, scales=[0],\
                robust=1.0, uvtaper='', niter=0, gain=0.1, nsigma=2.0, \
                cycleniter=5000, cyclefactor=3, mask='', savemodel="none", calcres=True,\
                calcpsf=True, parallel=True   ):
    if mask:  mask = imagename_base+mask
    tclean(vis=vis, field=field, spw=spw, uvrange=uvrange, datacolumn=datacolumn,\
           imagename=imagename_base+image_iter, imsize=imsize, \
           antenna=antenna, scan=scan, intent=intent, \
           cell=cell, phasecenter=phasecenter, reffreq=reffreq, gridder="awproject",\
           wprojplanes=wprojplanes, cfcache=cfcache, conjbeams=True, \
          usepointing=usepointing, rotatepastep=5.0, pblimit=0.02, deconvolver="mtmfs",\
          scales=scales, nterms=2, smallscalebias=0.4, weighting="briggs",\
          robust=robust, uvtaper=uvtaper, niter=niter, gain=gain, threshold=0.0,\
          nsigma=nsigma, cycleniter=cycleniter, cyclefactor=cyclefactor, usemask="user",\
          mask=mask, restart=True, savemodel=savemodel, calcres=calcres, calcpsf=calcpsf,\
          parallel=parallel)


def replace_psf(old,new):
    for this_tt in ['tt0','tt1','tt2']:
        shutil.rmtree(imagename_base+old+'.psf.'+this_tt)
        shutil.copytree(imagename_base+new+'.psf.'+this_tt, imagename_base+old+'.psf.'+this_tt)
    shutil.rmtree(imagename_base+old+'.workdirectory')


def replace_rflag_levels():
    for infile,indev in zip(['fdev.txt','tdev.txt'],['freqdev','timedev']):
        with open(infile,'r') as in1:
            instring = in1.readlines()
            dev = eval( instring[0] )
        dev_median = str(np.median(np.array(dev[indev])))
        pattern=re.compile('(\[[\d\.]*,[\d\.]*,)([\d.]*)(\])')
        s=re.sub(pattern,'\\1bla\\3',instring[0])
        s2=re.sub('bla',str(dev_median),s)
        with open(infile,'w') as out1:
            out1.write(s2)


#Steps are added in comments that refer to the VIP workflow steps outlined in Section 3.

# Step 3: create uniform psfs with wbawp=False
run_tclean( 'iter0', cfcache=cfcache_nowb, robust=-2.0, uvtaper='3arcsec', calcres=False  )


# Step 3: create robust psfs with wbawp=False
run_tclean( 'iter0b', cfcache=cfcache_nowb, calcres=False  )


# Step 4: make mask from QL catalog
mask_from_catalog(inext=inext,outext="QLcatmask.mask",catalog_search_size=1.5)


# Step 5: initialize iter1, no cleaning
run_tclean( 'iter1', robust=-2.0, uvtaper="3arcsec"  )


# Step 5: replace iter1 psf with no_WBAP
replace_psf('iter1','iter0')


# Step 5: clean with mask down to 3 sigma using no scales, uniform weighting (further mitigates divergence)
# this is a temporary image used for flagging, statwt and selfcal
run_tclean( 'iter1', robust=-2.0, uvtaper="3arcsec", niter=20000, nsigma=5.0,\
            mask="QLcatmask.mask", calcres=False, calcpsf=False  )


# Step 5:  resume iter1 to save model
run_tclean( 'iter1', calcres=False, calcpsf=False, savemodel='modelcolumn', parallel=False  )


# Step 6:  RFI Flagging; node that the pipeline should not adopt
#          the heuristics utilized below, these were used to avoid dependence
#          on the pipeline for VIP.
flagdata(vis=vis, mode='rflag', datacolumn='residual_data',timedev='tdev.txt',\
         freqdev='fdev.txt',action='calculate')
replace_rflag_levels()
flagdata(vis=vis, mode='rflag', datacolumn='residual_data',timedev='tdev.txt',\
         freqdev='fdev.txt',action='apply',extendflags=False)
flagdata(vis=vis, mode='extend', extendpols=True, growaround=True)

#Step 7: Run statwt in channelized mode on residual_data
statwt(vis=vis,combine='field,scan,state,corr',chanbin=1,timebin='1yr',\
       datacolumn='residual_data' )


# Step 8: selfcal
gaincal(vis=vis,caltable='g.0',gaintype='G',calmode='p',refant='0',combine='field,spw',minsnr=5)
applycal(vis=vis,calwt=False,applymode='calonly',gaintable='g.0',spwmap=18*[2], interp='nearest')


# Step 9: create uniform psfs with wbawp=False using corrected column
run_tclean( 'iter0c', datacolumn='corrected', cfcache=cfcache_nowb, robust=-2.0,\
            uvtaper='3arcsec', calcres=False  )


# Step 9: create robust psfs with wbawp=False using corrected column
run_tclean( 'iter0d', datacolumn='corrected', cfcache=cfcache_nowb, calcres=False  )
```

```
# Step 10: initialize iter1b, no cleaning, using corrected column
run_tclean( 'iter1b', datacolumn='corrected', robust=-2.0, uvtaper="3arcsec" )


# Step 10: replace iter1b psf with no_WBAP
replace_psf('iter1b','iter0c')


# Step 10: clean with mask down to 3 sigma using no scales, uniform weighting, using corrected column
# this is a temporary image used for Tier-2 masking
run_tclean( 'iter1b', datacolumn='corrected', robust=-2.0, uvtaper="3arcsec", niter=20000,\
           nsigma=5.0, mask="QLcatmask.mask", calcres=False, calcpsf=False  )


# Step 11: make Tier-2 mask from iter1 image after smoothing
imsmooth(imagename=imagename_base+"iter1b.image.tt0", major='5arcsec', minor='5arcsec',\
         pa='0deg', outfile=imagename_base+"iter1b.image.smooth5.tt0")
exportfits(imagename=imagename_base+"iter1b.image.smooth5.tt0",\
           fitsimage=imagename_base+"iter1b.image.smooth5.fits")
subprocess.call(['/users/jmarvil/scripts/run_bdsf.py',\
                 imagename_base+'iter1b.image.smooth5.fits'],env={'PYTHONPATH':''})


# Step 11: turn 5" smooth results back into component catalog and then to mask, using artifact rejection
edit_pybdsf_islands(catalog_fits_file=imagename_base+'iter1b.image.smooth5.cat.fits')
mask_from_catalog(inext=inext,outext="secondmask.mask",\
                  catalog_fits_file=imagename_base+'iter1b.image.smooth5.cat.edited.fits',\
                  catalog_search_size=1.5)


# Step 12: combine Tier-1 and Tier-2 order masks
immath(imagename=[imagename_base+'secondmask.mask',imagename_base+'QLcatmask.mask'],\
       expr='IM0+IM1',outfile=imagename_base+'sum_of_masks.mask')
im.mask(image=imagename_base+'sum_of_masks.mask',mask=imagename_base+'combined.mask',\
        threshold=0.5)


# Step 13: initialize iter2, no cleaning
run_tclean( 'iter2', datacolumn='corrected' )


# Step 13: replace iter2 psf with no_WBAP
replace_psf('iter2','iter0d')


# Step 13: resume iter2 with QL mask
run_tclean( 'iter2', datacolumn='corrected', scales=[0,5,12], nsigma=3.0, niter=20000,\
            cycleniter=3000, mask="QLcatmask.mask", calcres=False, calcpsf=False  )


# Step 14: iter2 with combined mask
os.system('rm -rf *.workdirectory')
os.mkdir('iter2_intermediate_results')
os.system('cp -r *iter2* iter2_intermediate_results')
shutil.rmtree(imagename_base+'iter2.mask')
shutil.copytree(imagename_base+'combined.mask',imagename_base+'iter2.mask')
run_tclean( 'iter2', datacolumn='corrected', scales=[0,5,12], nsigma=3.0, niter=20000,\
            cycleniter=3000, mask="", calcres=False, calcpsf=False  )



os.system('rm -rf *.workdirectory')
```

Revision History

Revision	**Date**	**Author(s)**	**Description**
1.0	2020-05-07	J. Tobin, J. Marvil, J. Ott., S. Bhatnagar	Original