River Detection algorithm

Documentation

INTRODUCTION

# About The Algorithm

We have built an image processing algorithm in python 3, which takes satellite river image as an input, analyses it, and returns data about the features of the river like the information about the different reaches, junctions, isolated land areas and most importantly the average width of all the reaches and also the river.

The code is mainly divided into two parts:

1. Image Preprocessing
2. Calculating the average width

The algorithm uses complex libraries and algorithms like Otsu’s binarization for thresholding, contour recognition for finding linear river geometries, skeletonization and a few more. We will be discussing the whole algorithm in detail as we go ahead.

# Libraries Used

The python libraries used are as follows:

1. Cycler
2. Decorator
3. Image
4. Kiwisolver
5. Matplotlib
6. Mysql-connector-python
7. Networkx
8. Numpy
9. Opencv-python
10. Pillow
11. Protobuf
12. Pyparsing
13. Python-dateutil
14. Pytz
15. PyWavelets
16. Scikit-image
17. Scipy
18. Six

HOW DOES THE CODE WORK?

# Image Preprocessing

The algorithm is split into two main sections, here we discuss the image preprocessing.

1. **Taking the input**

The code takes satellite image as an input and converts it into 8-bit greyscale format. Where each pixel has an intensity, ranging from 0-255. This helps the code to distinguish the river from the land, as the river is darker than the land, when converted into greyscale format.

1. **Thresholding the image**

To threshold the image we first find a threshold value, this is done using the Otsu’s thresholding algorithm. The algorithm is explained below.

Otsu’s thresholding method involves iterating through all the possible threshold values and calculating a measure of spread for the pixel levels each side of the threshold, i.e. the pixels that either fall in the foreground or background. The aim is to find the threshold value where the sum of the foreground and background spreads is the minimum. For better understanding the algorithm, links will be provided in the reference section.

Once we find the threshold value, we can now threshold the image. This gives us a binary image with the land painted white and the river painted black.

1. **Rotating the image**

To rotate the image, we have to first identify the river and all its tributaries, and also their angle of rotations. We do this using contours and minimum area bounding rectangles.

We now identify all the contours in the image, but the river and tributaries mostly have linear geometries. Hence, we use minimum area rectangles, which tend to bound these river contours by occupying minimum area (these rectangles may have some angle of rotation). As a result, we find all the river patterns and their angle of rotations.

We now find the median angle of rotation and use this angle to rotate the image, in order to straighten it.

1. **Cropping the image**

But Rotation introduces some new area which is also white in color (same as the river), this happens because every image has to maintain its rectangular dimensions.

So, we crop the image using two points namely the left-uppermost and the right-bottommost points. To obtain these we again find all the contours in the image but now we bound them using straight rectangles. These rectangles help us in finding the topmost and the leftmost contours. Now we find the (x,y) coordinates of the left-uppermost point using the x-coordinate of the leftmost contour bounding rectangle and the y-coordinate of the topmost contour bounding rectangle, we similarly do this for the right-bottommost point also. And using these two points we crop the image. This helps eliminate some of the rotation introduced area.

1. **Inverting colors and applying a black boundary**

We invert the colors of the image using the threshold function again but this time passing the invert parameter.

And then we apply a boundary to the rotated image. The algorithm for this works as follow. First, we select a corner – say the upper-left corner. Then we travel down from this corner till we find a black pixel, similarly we travel to the right from this corner till we find another black pixel we then draw a line between these two points. This is repeated over all the corners and finally we obtain a bounded image.

1. **Eliminating the outer areas**

As of now the river and the outer areas are both in white, so we have to paint the outer areas black.

This is done using the flood fill function. We use flood fill on all the four corners of the image and hence, eliminate the outer areas.

1. **Choosing a seed point**

A seed point is a point that is inside the river. The algorithm for this works as follows.

We have with us the binary image which contains the river and some isolated water patches in white and the land in black. So, the algorithm iteratively chooses white pixels and flood fills them, while keeping track of the pixel which flood fills the maximum area. Because this maximum will always be the river.

1. **Removing isolated water patches and painting the river blue**

Now we use the seed point found above to flood fill the river blue. But there are still some water patches in the image in white color. So, we again iteratively choose white pixels but now we flood fill them black.

This helps us get a clean image with only the river and its tributaries painted blue and all other region painted black.

1. **Finding isolated land areas**

To obtain the isolated land areas we use a copy of the above image. The algorithm uses a basic fact that the isolated land areas are surrounded by the river and hence flood filling the outer areas blue using all four corners of the image won’t hurt. Once this is done we get the isolated land areas in black and rest everything in blue. Now we segment this image with blue lines as per the interval provided by the user (dividing the image from top to bottom into sections of user defined length). Iteratively identifying these black regions, we store the information like number of land areas in each section and the number of pixels they contain.

1. **Finding the skeleton**

The skeleton of the river is found using the – morphology.skeletonize() function. The algorithm works as follows.

It takes the river and reduces it from both the sides until a single pixel wide skeleton is left. This process is also called as thinning. More information on this function will be provided in the reference section.

1. **Removing dangling arcs**

The skeleton found may have many dangling arcs which must be removed. This is achieved by traversing all the arcs in the skeleton. If we encounter any arc which is less than the permissible length (length provided by the user) we delete that arc.

1. **Finding the junctions**

We now move towards finding the junction points. The algorithm used here is simple.

We iteratively check for all the skeleton points that have 3 or more branches emerging from them. These points are then stored as a list of junction type variables where each point has a unique junction-id.

1. **Finding the reaches and stream points**

A reach is a part of the skeleton which starts from a junction and may or may not end by one. The reaches are found by traversing all the branches emerging from the junction points. Each point of the reach is called a stream point. So, traversing all the reaches help us identify all the stream points. Now that we have identified all the reaches, we make a list of the stream points they contain.

This concludes the image preprocessing section. Now we move on to the width calculation section.

# Calculating the average width

In this section we discuss about the width calculation.

1. **Calculating length of all the reaches**

This is done by iteratively traversing all the reaches and finding the Euclidean distance between all the adjacent stream points of that reach and hence summing all these distances to find the total length of the reach.

1. **Calculating width across all the stream points**

We first pick a reach and then one by one we pick all of its stream points. At each stream point (call it the center point) we identify a point before and a point after it, which both lie on the skeleton (call them support the points). We now use these support points to find the equation of a line perpendicular to the skeleton and which also passes through the center point.

Using this equation, we draw two lines, one on either side of the center point which are incident onto the river boundaries. We then calculate the length of these lines and add them to find the width across that stream point.

This is repeated for all the stream points of the given reach. But it might so happen that width may get calculated twice, because multiple skeleton points/branches emerging from a junction may not have a boundary between them. So, we ignore these regions by using a simple distance check. This is done by checking the absolute difference between the length of the perpendiculars drawn on either sides of the stream point. If this difference is less than 8 (can be set according to the user preference), we calculate the width across that stream point.

1. **Calculating average width of all the reaches**

We have the width across all the stream points calculated. We now iteratively pick a reach and find its average width, by summing the widths across all of its stream points and dividing it by the number of stream points. In this calculation we exclude the bad stream points identified in the check function stated above.

1. **Calculating average width of the river**

Finally, we calculate the average width of the river by summing the average widths of the reaches and dividing it by the number of reaches.

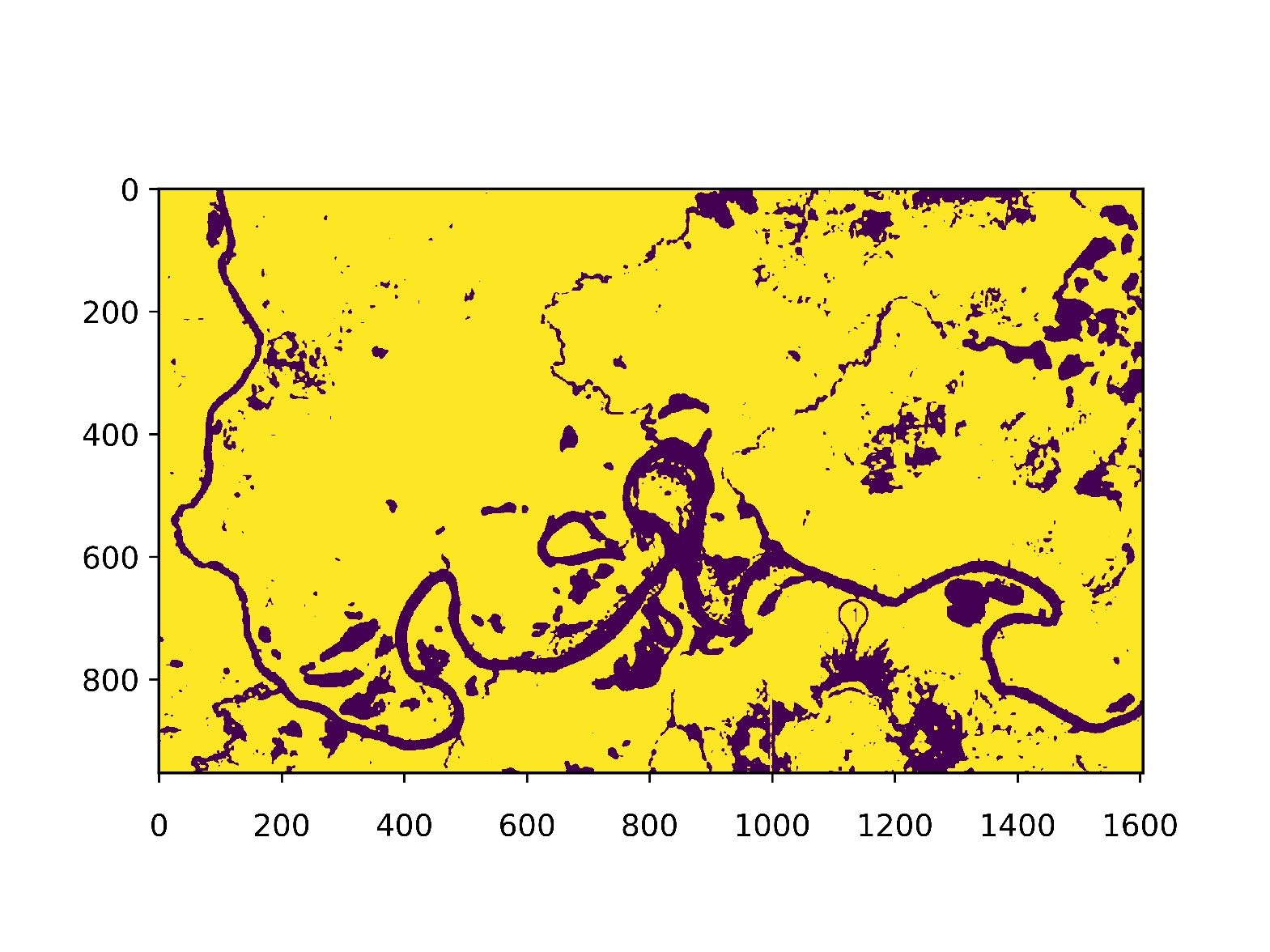
Here we are done with the width calculation section. This section marks an end to the code explanation. For more details about the functions and libraries used, please read the reference section.

IMAGES OF THE CODE AT WORK

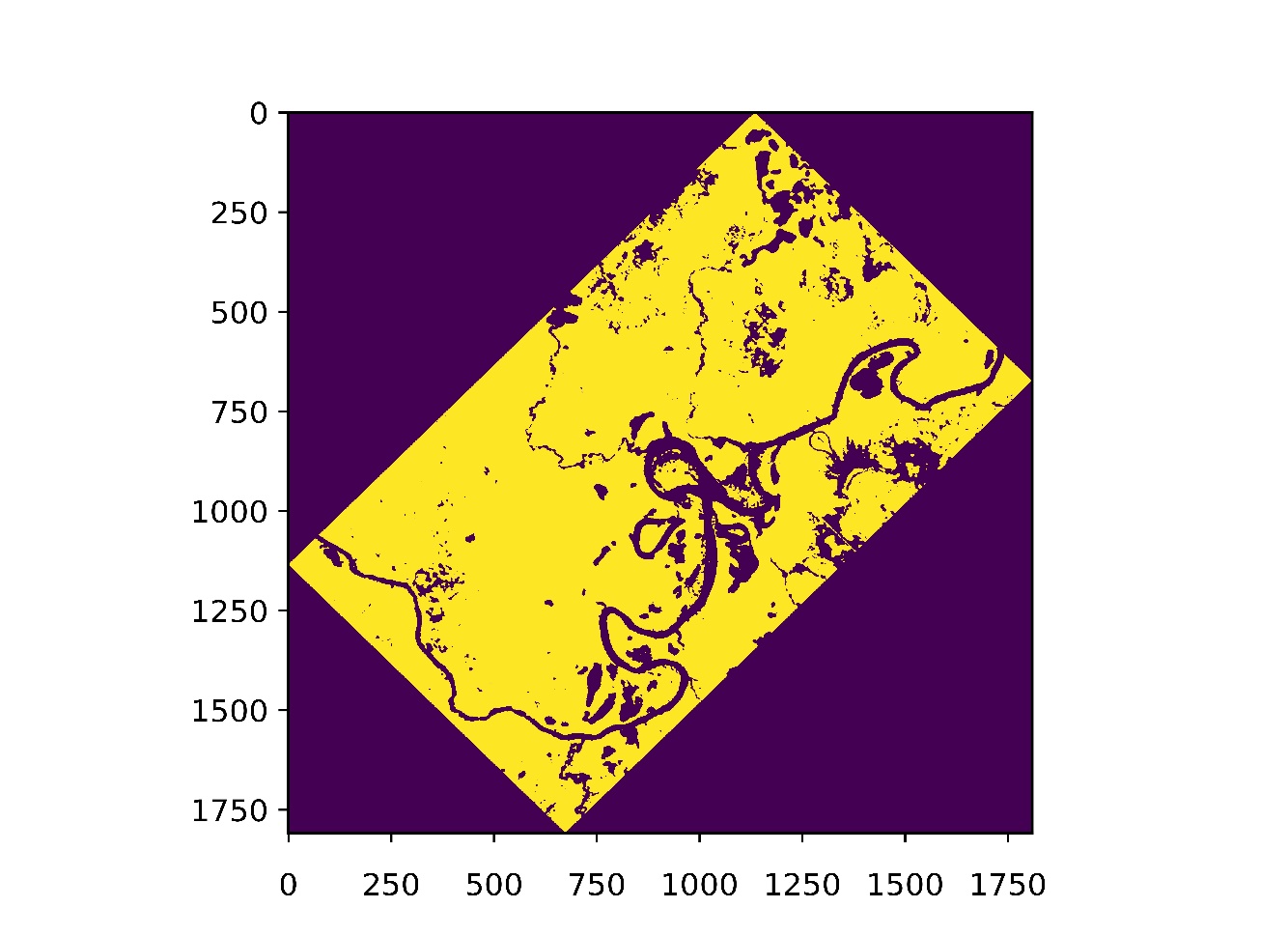
# Original Image



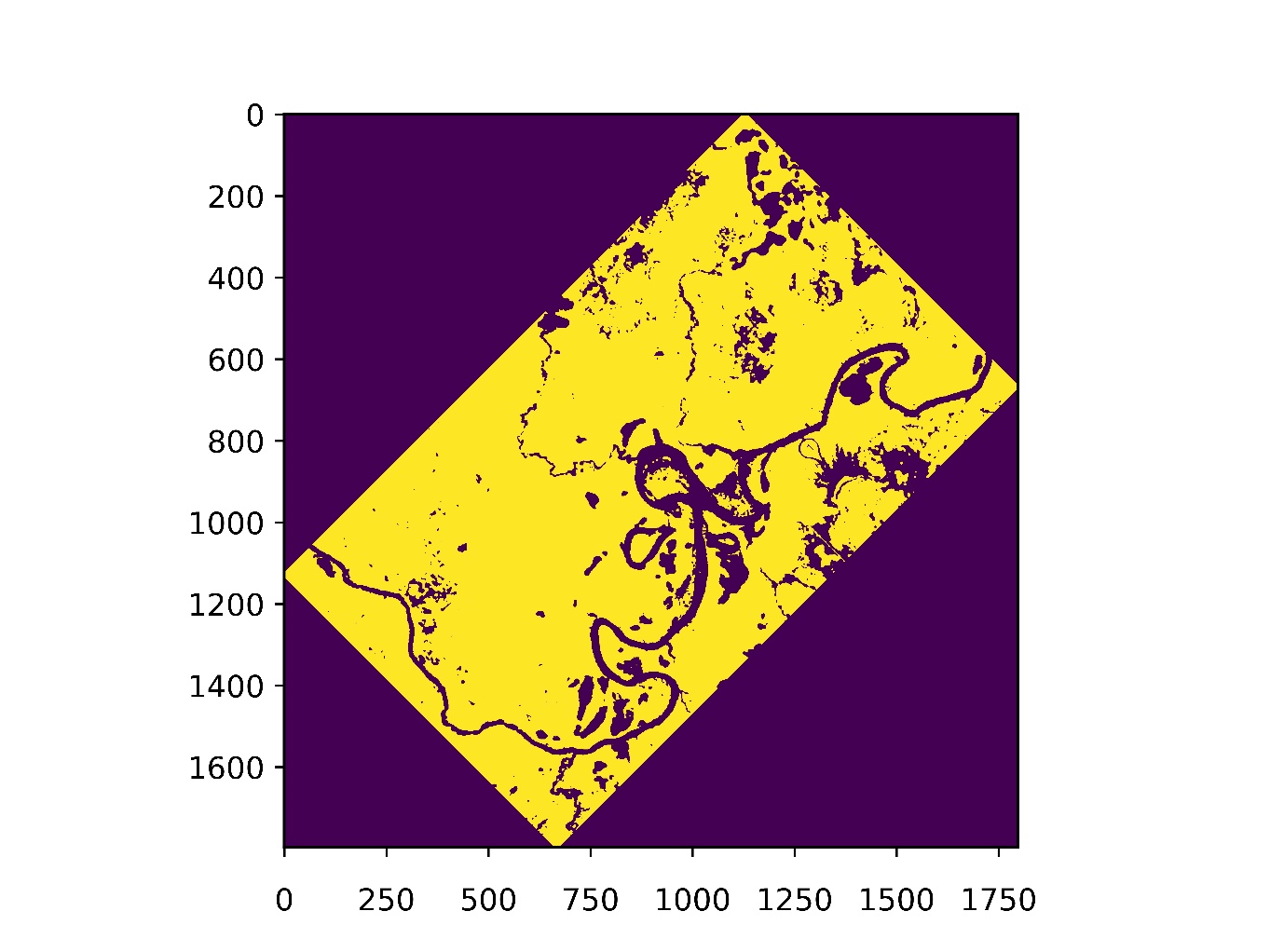
# Thresholding

****

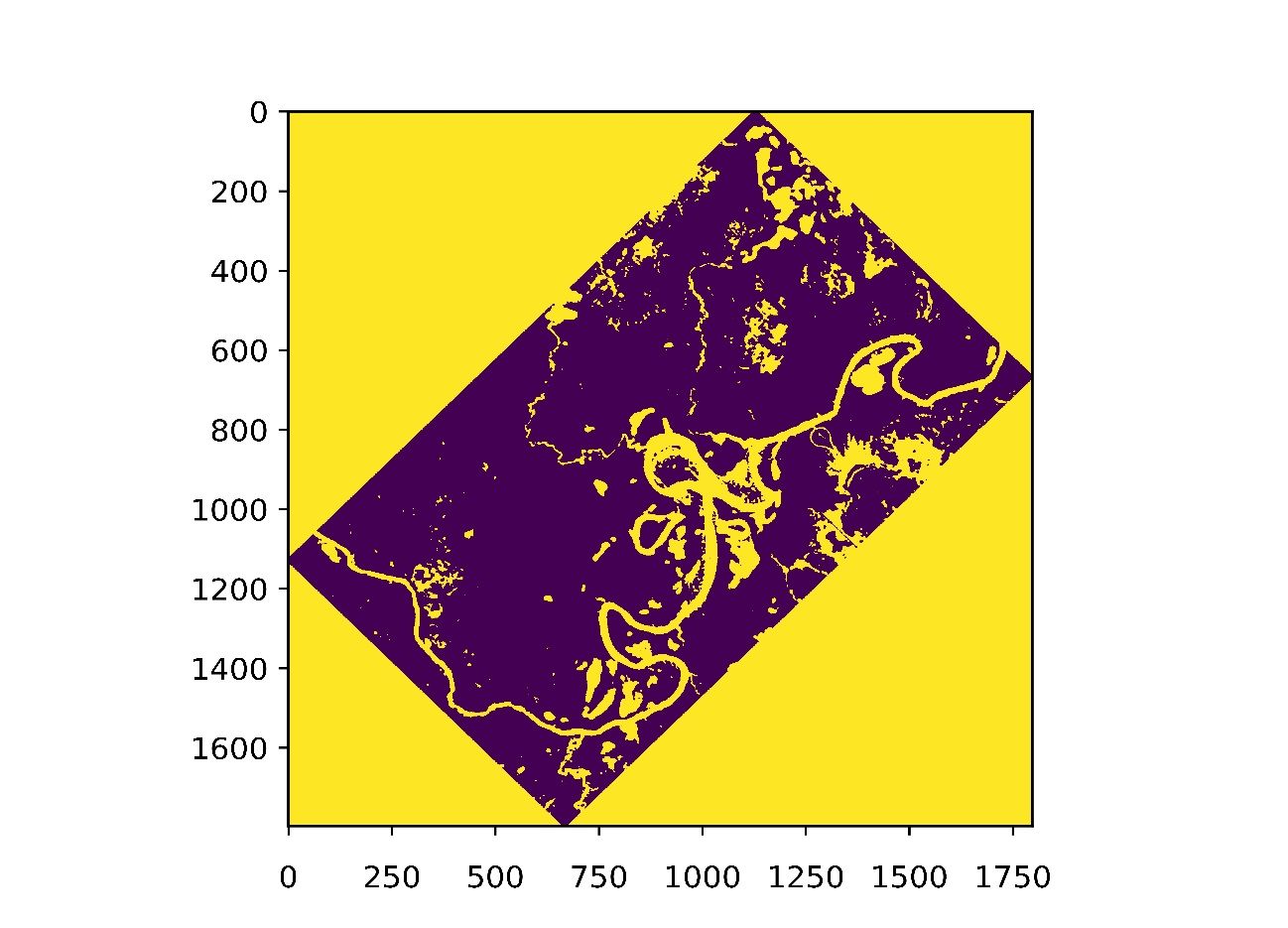
# Rotation

****

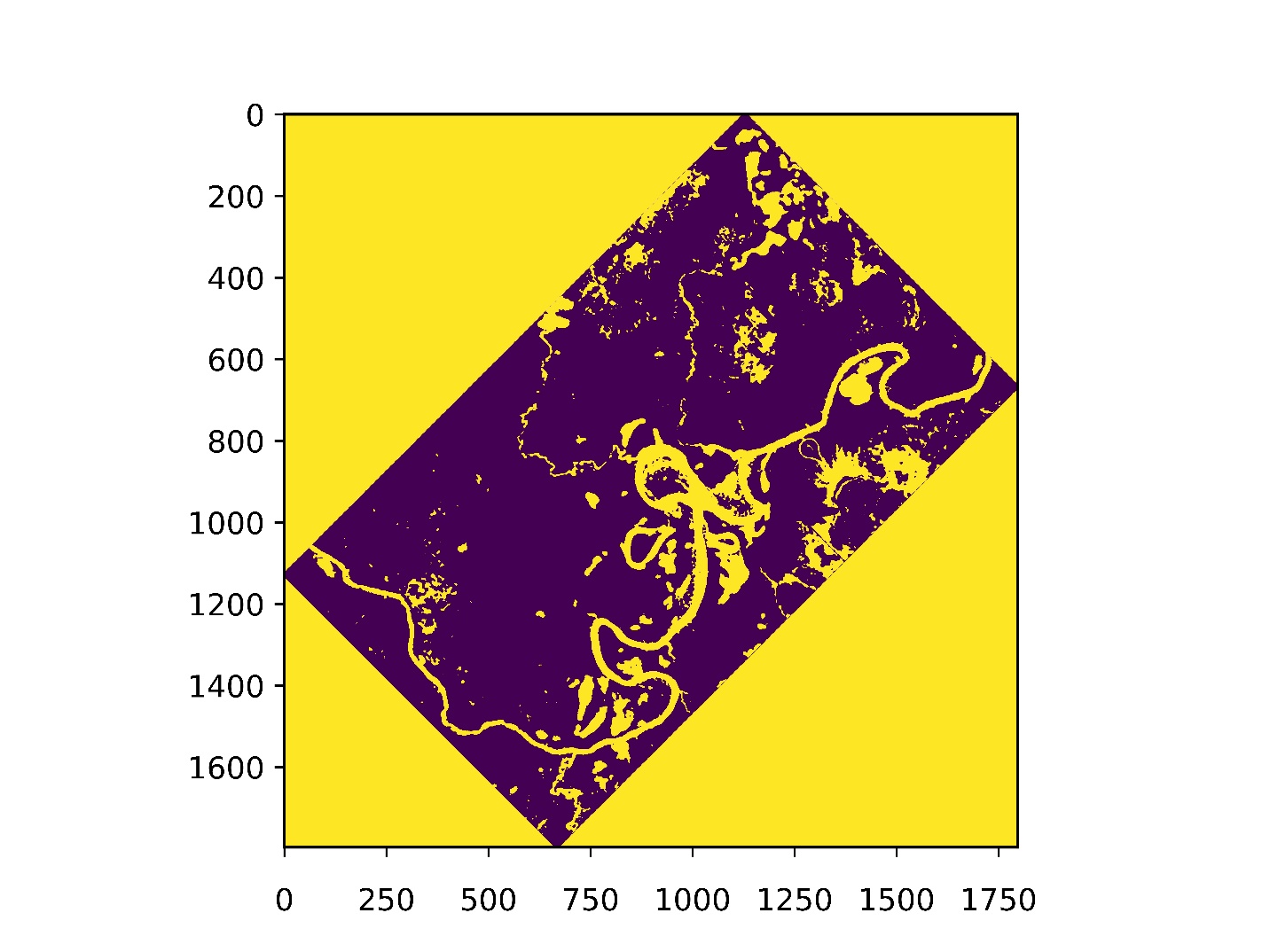
# Cropping

****

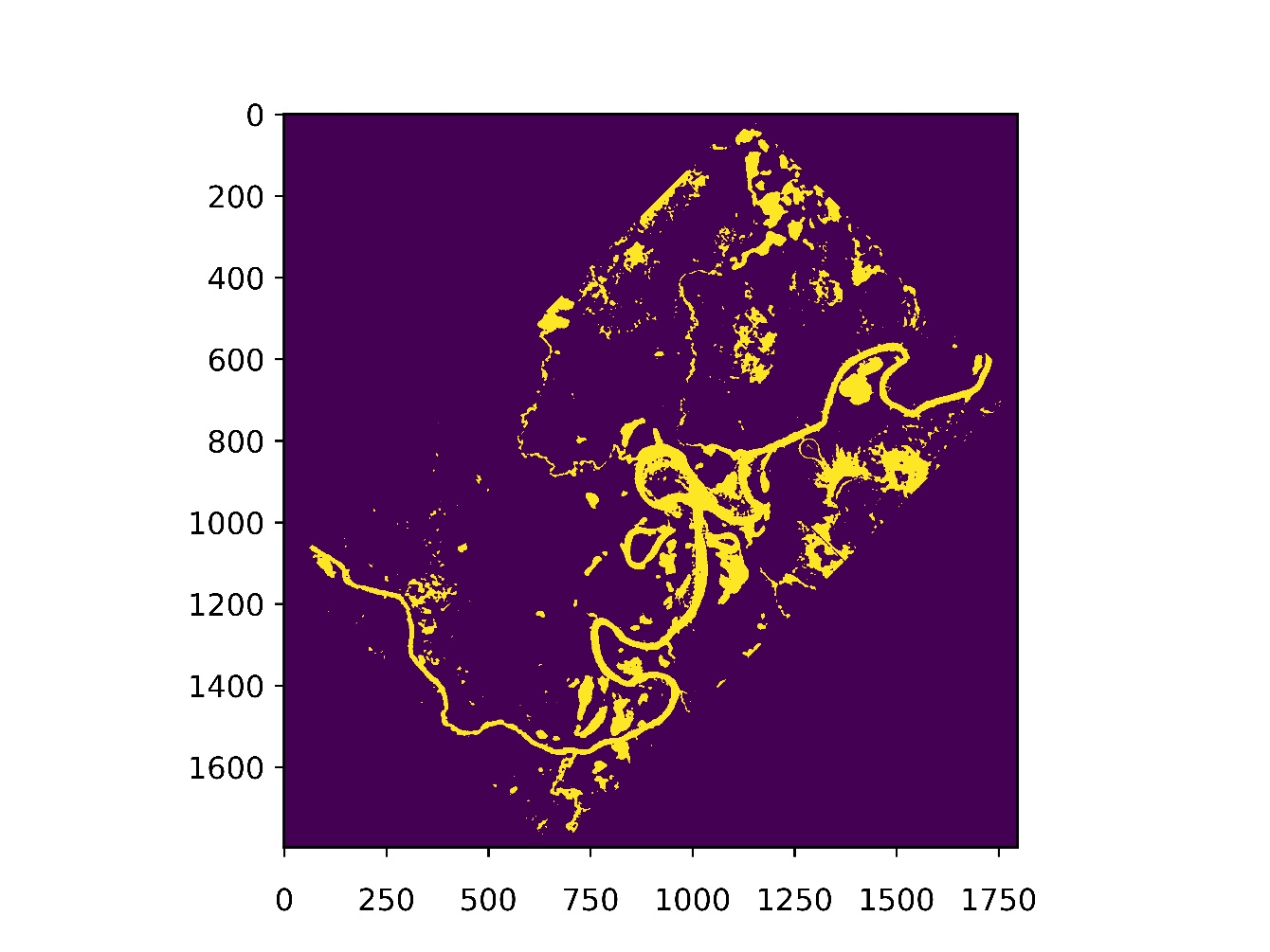
# Inverting the Colors

****

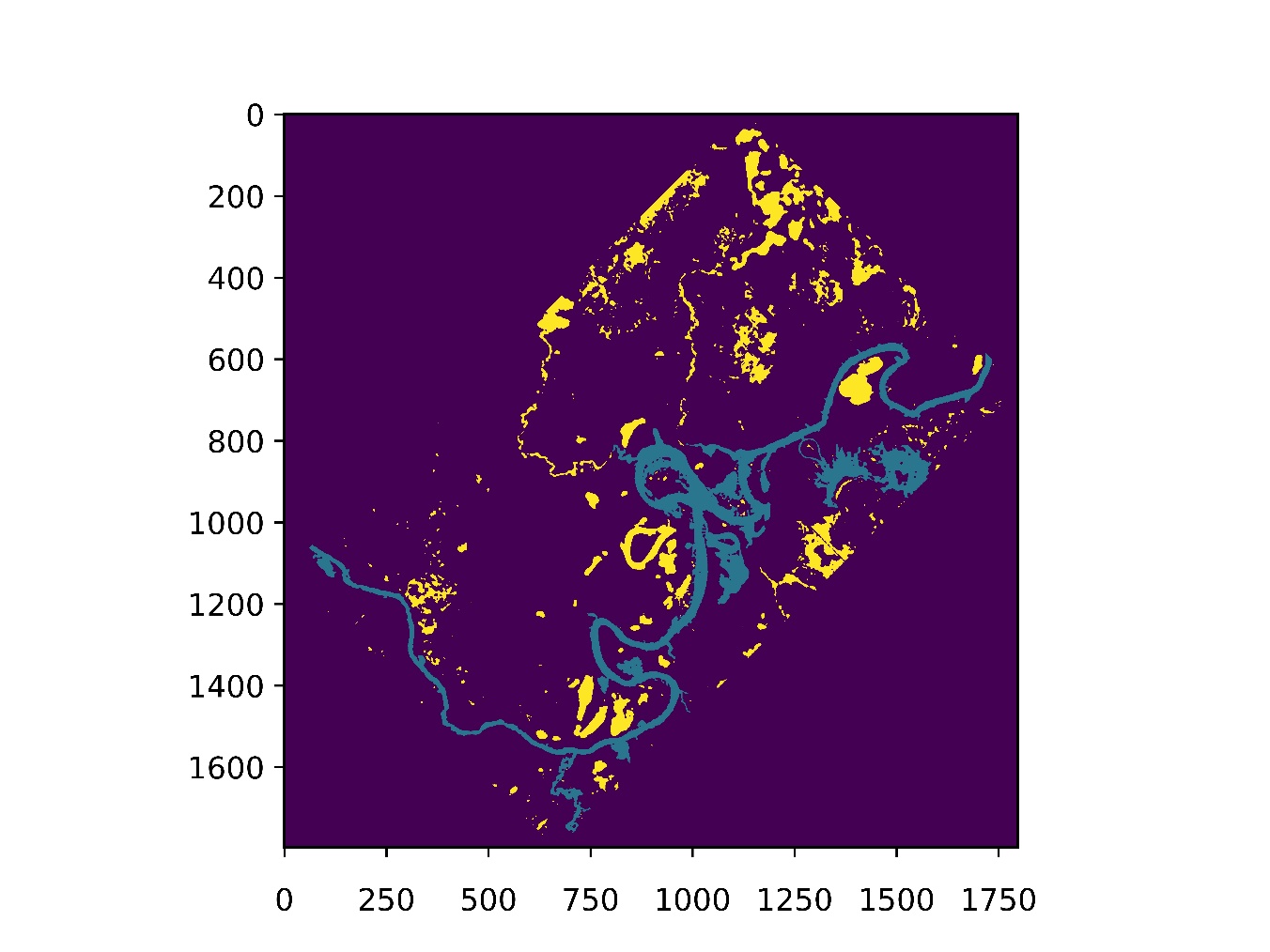
# Closing the Boundaries



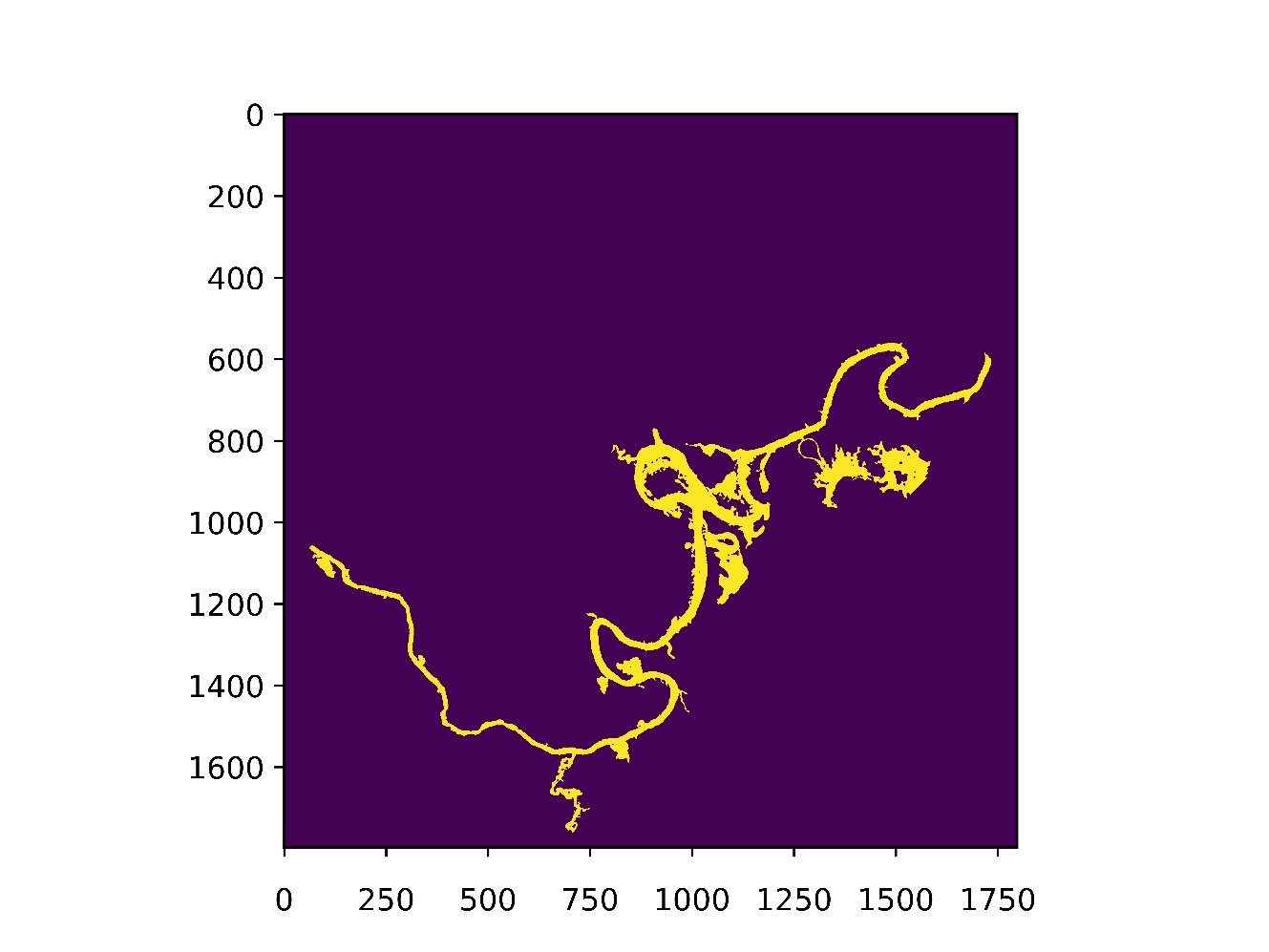
# Removing Outer Areas

****

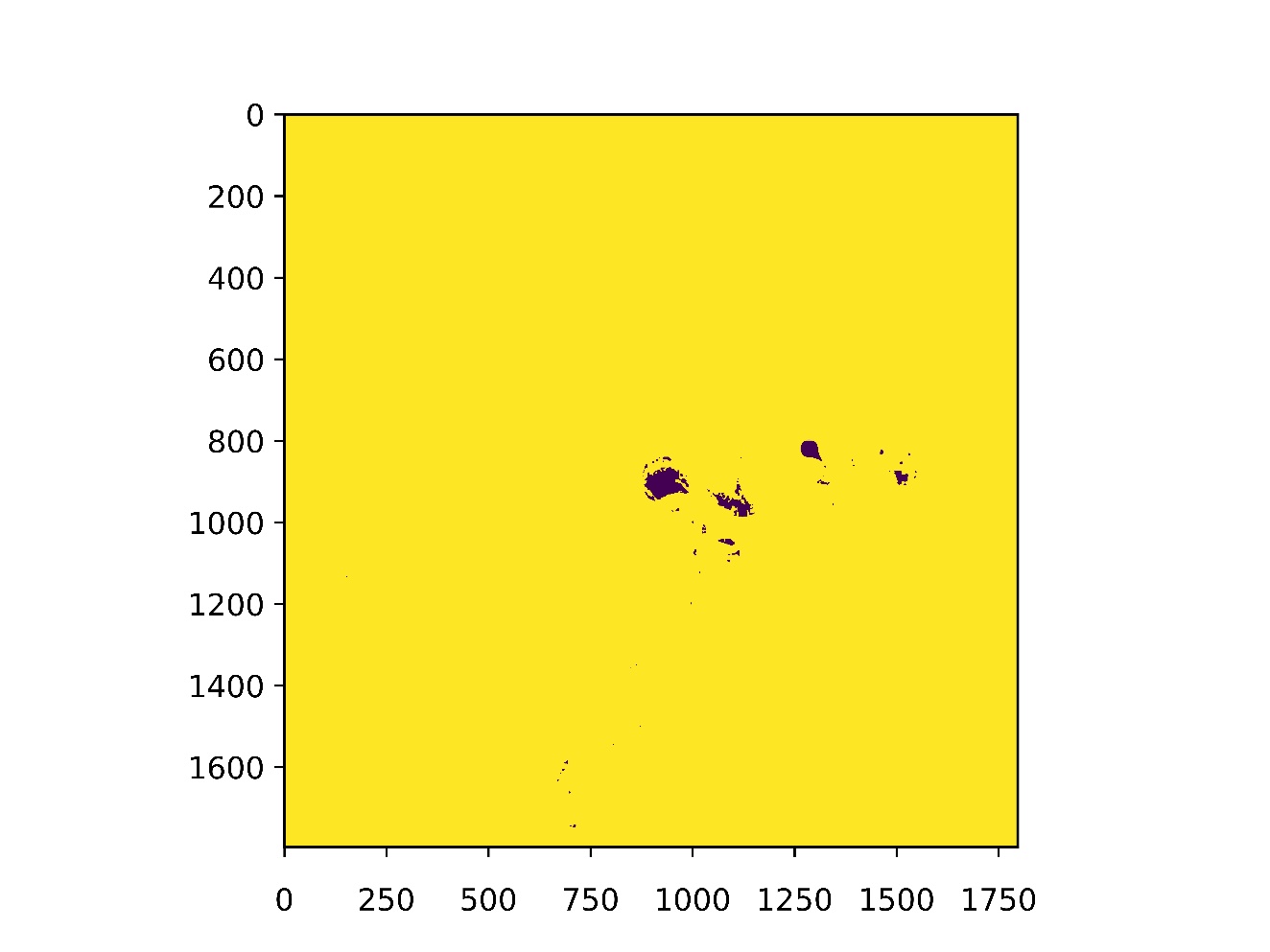
# Identifying River

****

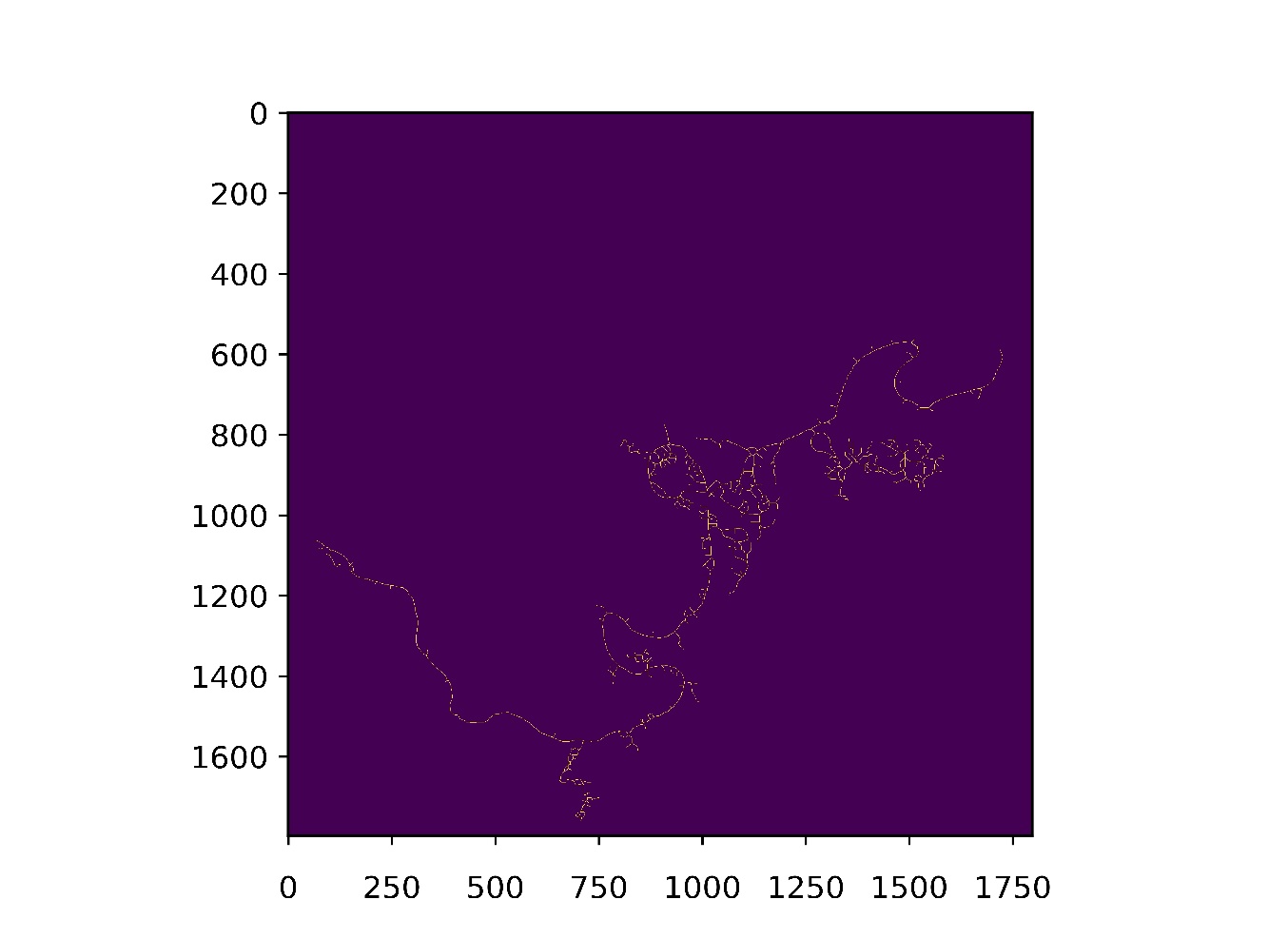
# Removing Isolated Patches of Water

****

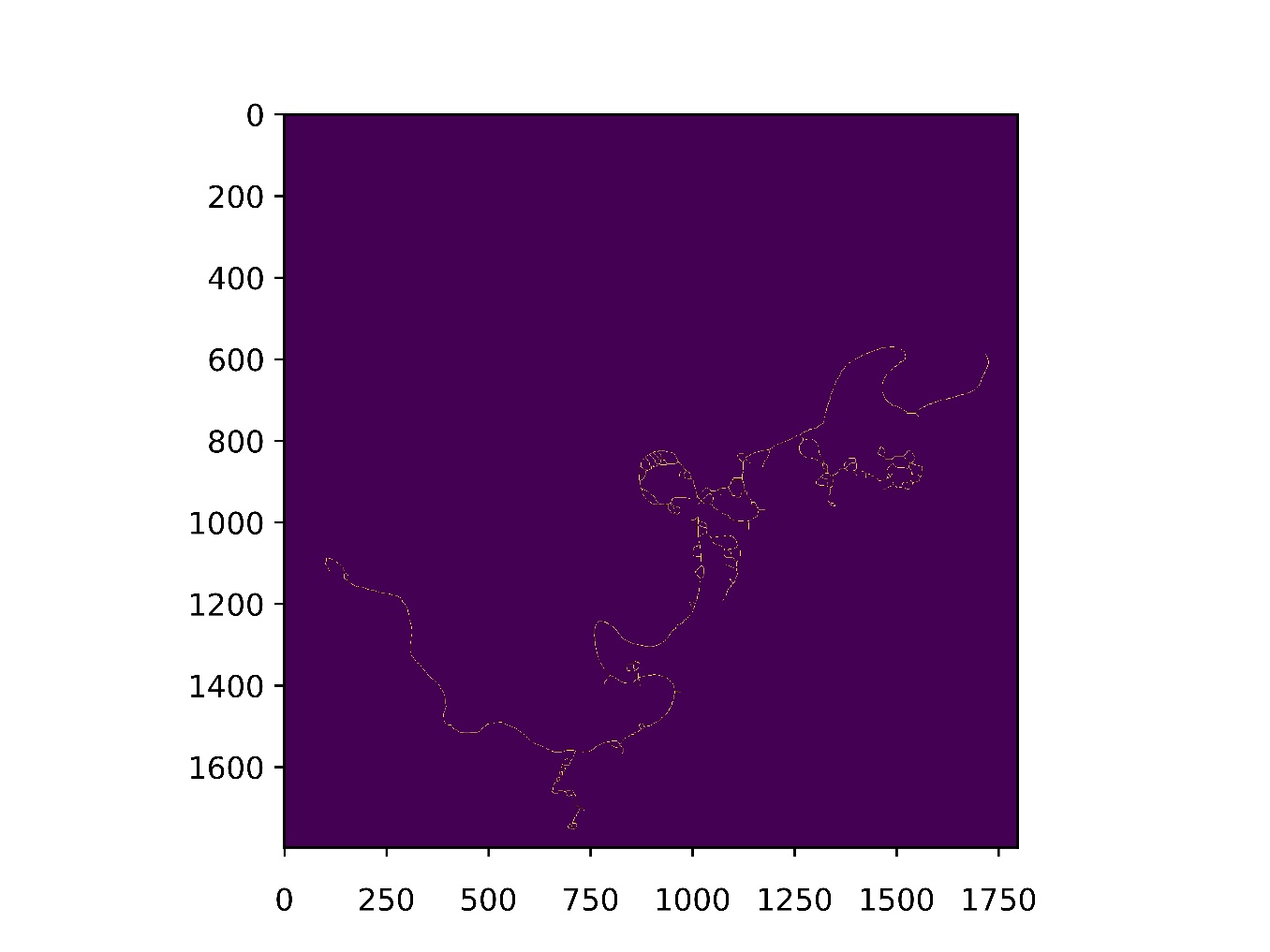
# Finding Isolated Patches of Land

****

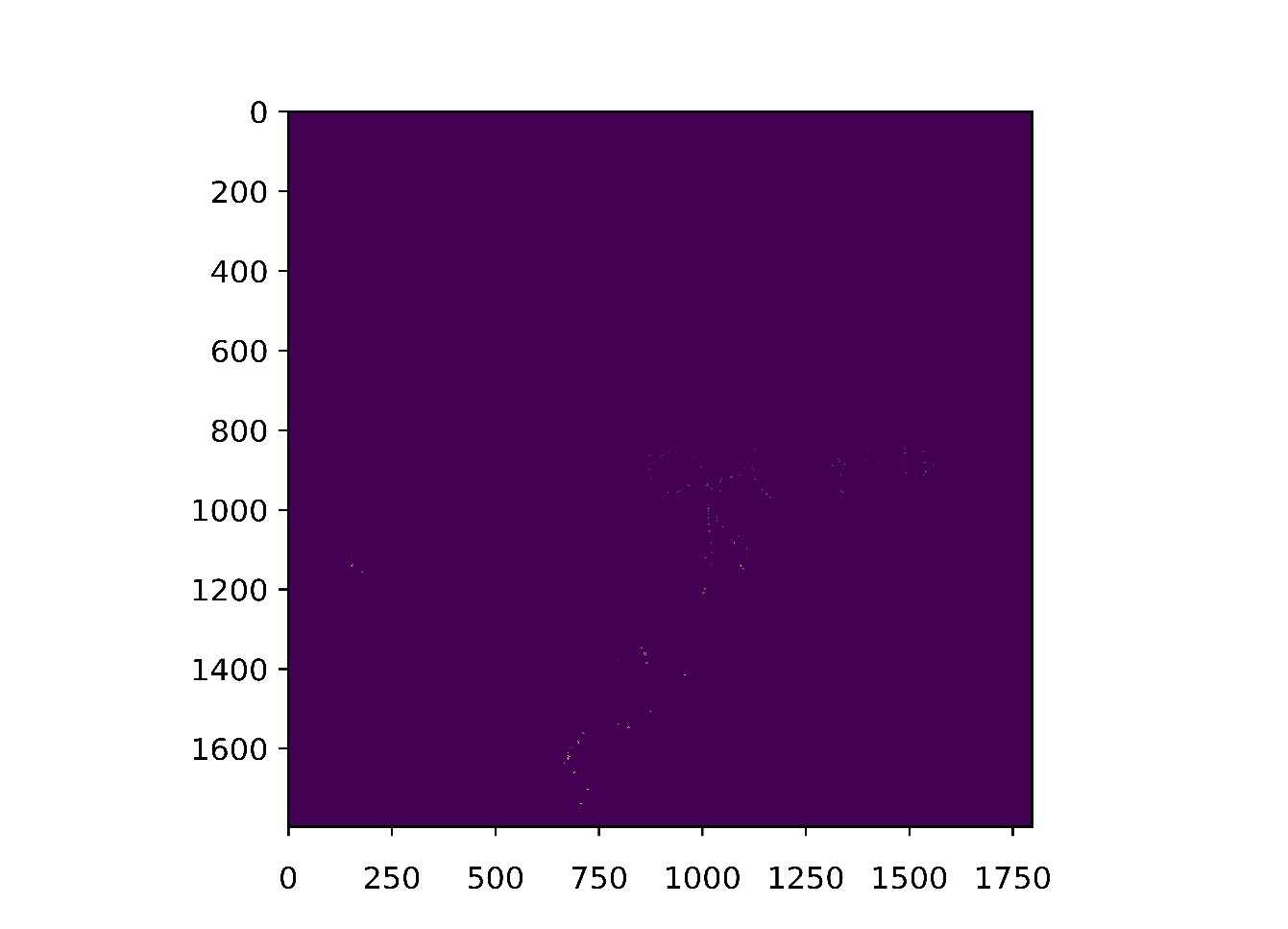
# Identifying Skeleton

****

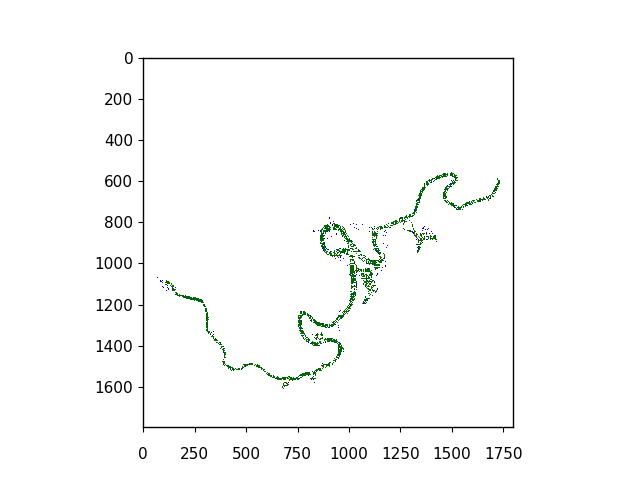
# Removing Dangling Arcs

****

# Identifying Junctions

****

# Final Result

****

REFERENCE LINKS

# Functions

* [**cv2.imread**](https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_gui/py_image_display/py_image_display.html)
* [**cv2.GaussianBlur**](https://docs.opencv.org/3.1.0/d4/d13/tutorial_py_filtering.html)
* [**cv2.equalizeHist**](https://docs.opencv.org/3.1.0/d5/daf/tutorial_py_histogram_equalization.html)
* [**cv2.threshold**](https://docs.opencv.org/3.4.0/d7/d4d/tutorial_py_thresholding.html)
* [**cv2.findContours**](https://docs.opencv.org/3.1.0/d4/d73/tutorial_py_contours_begin.html)
* [**cv2.minAreaRect**](https://docs.opencv.org/3.1.0/dd/d49/tutorial_py_contour_features.html)
* [**cv2.floodFill**](https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html?)
* [**skimage.morphology.skeletonize**](http://scikit-image.org/docs/dev/api/skimage.morphology.html#skimage.morphology.skeletonize)
* [**matplotlib.pyplot.figure**](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.figure.html)
* [**matplotlib.pyplot.imshow**](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.imshow.html)
* [**matplotlib.pyplot.show**](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.show.html)

# Related Algorithms and Methods

* [**Otsu’s Algorithm for Image Binarization**](http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html)
* [**Global Thresholding Algorithm**](https://www.mathworks.com/matlabcentral/fileexchange/38390-basic-global-thresholding)
* [**Minimum Error Thresholding Algorithm**](https://www.ppgia.pucpr.br/~facon/Binarizacao/1986Kittler.pdf)