ECOSTRESS Geolocation Python Batch: Documentation

ECOSTRESS Document

This document explains the mechanics of the ECOSTRESS Geolocation Jupyter Notebook

Table of Contents

[ECOSTRESS Geolocation Accuracy 1](#_Toc193100583)

[Where to obtain ECOSTRESS Data? 2](#_Toc193100584)

[Necessary File(s) 2](#_Toc193100585)

[What is this Code’s Approach? 2](#_Toc193100586)

[Edge Detection 2](#_Toc193100587)

[Edge Alignment 3](#_Toc193100588)

[Potential Errors: 3](#_Toc193100589)

[Python Batch Code 5](#_Toc193100590)

[Modules 5](#_Toc193100591)

[Gdal Errors 5](#_Toc193100592)

[Functions 6](#_Toc193100593)

[def Clip\_Parent\_Water\_Mask 6](#_Toc193100594)

[def Clipping\_LST\_and\_Water\_Mask 6](#_Toc193100595)

[def Sorted\_Shorelines 7](#_Toc193100596)

[def Endlines 8](#_Toc193100597)

[def MatchingOverlay 8](#_Toc193100598)

[def Matching\_Shift 8](#_Toc193100599)

[Shift\_in\_Progress 9](#_Toc193100600)

[Main 15](#_Toc193100601)

[Main function 15](#_Toc193100602)

[LST Flag Comparison 15](#_Toc193100603)

[Running Main 16](#_Toc193100604)

# ECOSTRESS Geolocation Accuracy

ECOSTRESS relies on information from the International Space Station (ISS) for geolocation coordinates, which could be off by as much as 7km. Within ECOSTRESS geolocation processing, image matching is used to improve geolocation, although this is not always successful. ECOSTRESS scenes located near bodies of water have noticeable errors as observers can easily find discrepancies between a base map and the ECOSTRESS scene. This code aims to use ECOSTRESS’s water mask product to correctly place the ECOSTRESS file.

# Where to obtain ECOSTRESS Data?

ECOSTRESS data is available on AppEEARS which requires an account to download images. Please refer to JPL’s ECOSTRESS Tutorials ECOSTRESS Tutorial [06-Downloading\_From\_AppEEARS](https://github.com/ECOSTRESS-Tutorials/ECOSTRESS-Getting-Started/blob/main/06-Downloading_from_AppEEARS.md) for steps on setting up your account and downloading data. This code primarily works best with Collection 2 data as Collection 2 provides personalized water masks for each scene. AppEEARS provides each of the necessary files to a bounding box based on the users Region of Interest (ROI).

# Necessary File(s)

This code was built to process ECOSTRESS Collection 2 from [AppEEARS](https://appeears.earthdatacloud.nasa.gov/). There are two versions of ECOSTRESS LST Collection 2 data with Swath and Tiled. To obtain Swath tiled products you may search for the product: “ECOSTRESS Swath Land Surface Temperature and Emissivity Instantaneous” or search for the tiled product using: “ECOSTRESS Tiled Land Surface Temperature and Emissivity Instantaneous”. Below are the essential files with an optional cloud mask for additional edge removals.

* Land Surface Temperature (LST)
* Quality Control (QC)
* Water Mask
* Cloud Mask **(optional)**

Note: Running the python batch code with the single water mask option for a folder that has swath and tiled scenes with their native projections will result in errors. Please request your scenes from AppEEARS in any projection besides “Native”. The projection can be set under the “Projection: box under Output Options. If you plan on using Swath only or Tiled only, then this should not be an issue.

# What is this Code’s Approach?

This python batch code for ECOSTRESS Geolocation aims to match the ECOSTRESS LST shorelines with the ECOSTRESS water masks. The ECOSTRESS water masks are based on Shuttle Radar Topography Mission (SRTM) data which has higher geolocation accuracy and has been resized to ECOSTRESS’s 70-meter pixel resolution.

## Edge Detection

The main process to detect the shorelines of a scene is done by using canny edge detection via python’s *skimage* package. In most images, it is visually clear where the shoreline of each scene is based on your GIS program’s default color gradient and temperature values. Before edge detection is conducted, the image will have a gaussian blur applied to aid in identifying strong pixel differences and also to help reduce noise within the scene. Skimage’s canny edge detection identifies the most significant changes in values between pixels based on the gradient magnitude. The gradient magnitude is calculated by obtaining the pixels Sobel kernels for the x and y directions, compute the gradients on the images and finally applying the Pythagorean theorem using the resulting products. This process is shown below.

Sobel Kernels

Compute Gradients

Gradient Magnitude

## Edge Alignment

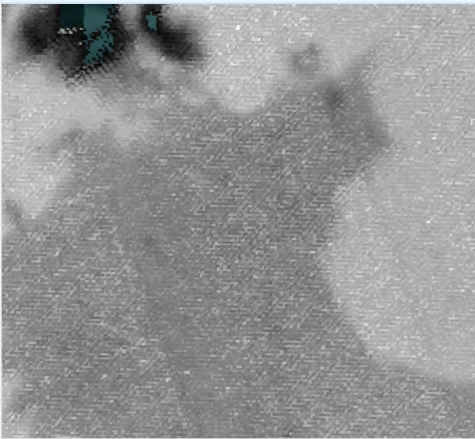
There is a process of hysteresis thresholding allowing for edges to be present based on if gradient values are either under, between, or above two thresholds. Automating this step is in development, as each image requires unique thresholds to isolate the LST shoreline. For this Jupyter Notebook version, no low or high thresholds will be set which will keep all edges found during edge detection.

After edge detection for the LST is computed, the edges go through a process of filtering to remove some unwanted edges. After edge filtration, the *best* edges are clipped by their own extended bounding box. The bounding box is expanded by 100 pixels in the x and y direction to match the 7km error. This bounding box will be used to clip the isolated ECOSTRESS edge and the associated Water Mask image. By rolling the ECOSTRESS image along the Water Mask, the best overlapping position is found and the shift is stored for calculation. For large scenes with multiple isolated shorelines, like Hawaii, each shoreline will be calculated individually. The final shift from every shoreline will be stored and used to find the best overall shift for the entire image.

## **Potential Errors**:

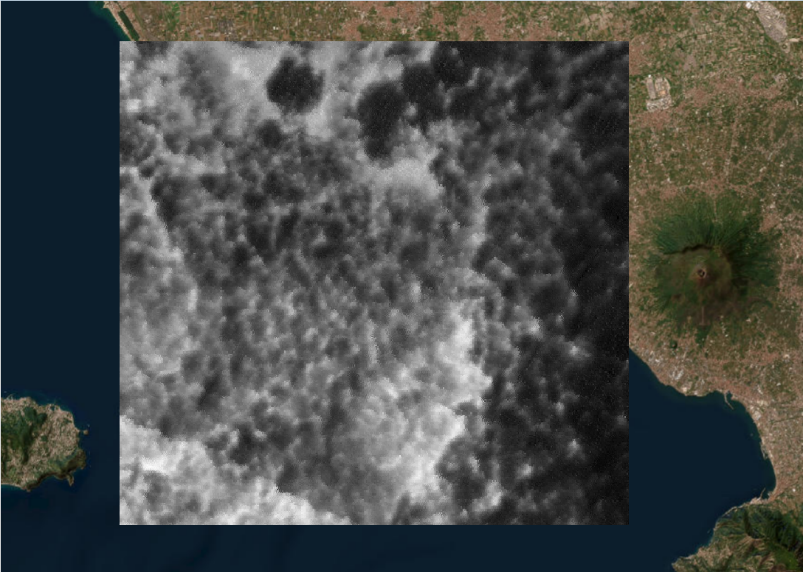
ECOSTRESS data, depending on the scene, may be noisy, cloudy, or have stripes in the LST image that may mask or breakup the shoreline present in the LST. Additionally, the water mask sometimes has issues where sections of the shoreline have been clipped resulting in suboptimal base map matching. If these scenes passed all set checks in the code, the image may be shifted incorrectly. Below are some visuals showing each potential error.

**Noisy Scenes**



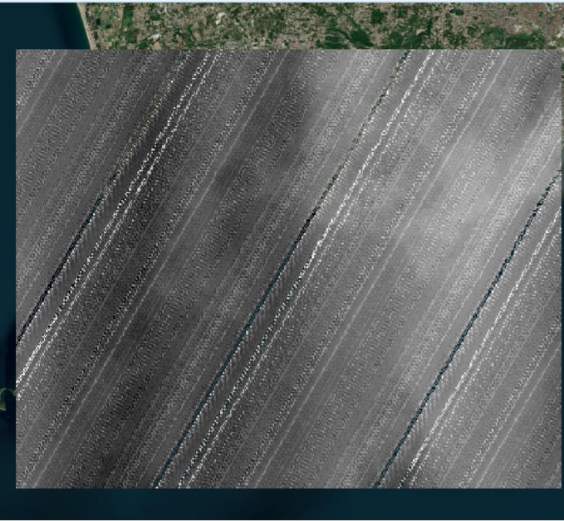
ECO\_L2T\_LSTE.002\_LST\_doy2023012222450\_aid0001\_30N

**Cloudy Scenes**



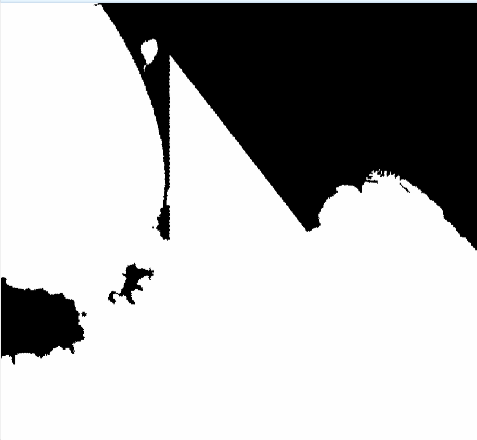
ECO\_L2\_LSTE.002\_doy\_2023360082254\_aid0001

**Image Striping**



ECO2LSTE.001\_SDS\_LST\_doy2023101144022\_aid0001

**Water Mask quality (Empty Segments)**

QGIS ESRI Imagery Base map ECO\_L2\_LSTE.002\_water\_mask\_doy2019040110918\_aid0001

For the best results, all LST images and water masks being processed should have 1) Connected shoreline, 2) Enough clear sky above the shoreline, and 3) an Ideal water mask. For the Jupyter notebook, the water mask will automatically be the one assigned to your specific LST. If the user prefers, the water mask may be a different water mask with a complete shore. The code will be able to read the “ideal” water mask and clip itself and the other files to the overlapping section between the “ideal” water mask and the LST image. If the LST does not overlap with the “ideal” water mask, then the LST image will not be corrected. Although this code works best with large shorelines, this code can work with inland bodies of water but may need to be large enough for matching to occur.

# Python Batch Code

The Python Batch version of the Geolocation code has similar steps as the Jupyter Notebook, however there are additional segments of code that help compute large folders. In the Python Batch version, the Jupyter Notebook is turned into a function as this version introduces multiprocessing requiring the code to be a function. This code also implements Geolocation Accuracy flags such as “Best”, “Good”,” Subject” and ”Bad”. These flags are being implemented into the supplementary statistics csv in each AppEEARS request, however, this code’s zipped folder includes geolocation flags from August 2018 - November 2024.

Additionally, This python code can be ran in 4 ways. These options are based on the users preference of using cloud masks or using a single water mask as their reference images. Cloud masks are a great option to remove any unwanted “LST” values however as shown in the images within the “Cloud mask on LST” section below, not all of the identified clouds in the cloud masks are clouds. There is a True or False variable that can be used to specify if the cloud mask will be used or not. This is one of the two variables used to make up the 4 different ways to use the code.

The second variable to use for matching is if the user wants to use a single water mask for the entire ROI or use individually customized water masks for each scene. As shown above in the potential errors section, the water mask has some inconsistencies in certain regions. Having this option is up to the user but by default the code will use a single water mask for your calculations. In future collections of ECOSTRESS, these inconsistencies will be addressed.

## Modules

import os

import re

import gc

import time

import cv2

import csv

import numpy as np

import concurrent.futures

from osgeo import gdal

from datetime import datetime

from pandas import DataFrame

from multiprocessing import freeze\_support

from scipy.ndimage import binary\_hit\_or\_miss

from skimage import feature, morphology, measure

These packages can be installed using the Geolocation.yml file located within this code’s GitHub. Additionally, please refer to this code’s [GitHub](https://github.com/Andrew-Alamillo/ECOSTRESS_Geolocation_Correction_tool) page for an explanation of each module as well to view the Python Batch User Guide.

## Gdal Errors

Gdal provides “future warnings” when gdal.UseExceptions or gdal.DontUseExceptions has not been specified. According to GDAL, version 4.0 will have exceptions enabled by default. As of writing this document, GDAL is in version 3.10. Due to this version, the following lines of code are added to remove any GDAL future warnings.

gdal.UseExceptions()

gdal.PushErrorHandler('CPLQuietErrorHandler')

## Functions

This part of the documentation explains the custom functions used in the code. Each function will be described in detail with descriptions for each input variable. Please refer to the Shift\_in\_Progress function for the main shifting processes. The entire code was placed into a function in order to run multiprocessing.

### def Clip\_Parent\_Water\_Mask

This function requires 5 inputs.

* **Parent\_Image**: Main water mask specified by the user
* **Temp\_Eco\_File:** The filename for the current ECOSTRESS image that is being shifted for
* **Temp\_Water\_File:** The water mask that is associated with the ECOSTRESS image
* **Temp\_QC\_File:** The QC file that is associated with the ECOSTRESS image
* **Temp\_Cloud\_File:** The cloud mask that is associated with the ECOSTRESS image

Note: Although this code has two inputs for water mask, this function will only return the parent water mask as this function will only be used if the user specifies a single water mask for their shift calculations.

The first step in this code is to open the Parent Image so that information such as its top right corner coordinates, height, width and resolution are extracted. From these values, the coordinates of the width and height based on the top right corner are found. Using the height and width of the parent image, there will be four zero matrices for the 4 necessary files. These zero matrices are created so every image has the same height and width. Having different array shapes will give errors when computing arrays.

After the zero matrices are created, the ECOSTESS image is opened and it’s information is extracted. By comparing the limits of both the Parent and ECOSTRESS scene, the coordinates of an overlapped segment can be calculated. These values are represented in the overlapped variables using max and min. Once that is calculated, the coordinates of the overlapped segments are compared to the resolution of each Parent and ECOSTRESS scene. These values are later used to only read the overlapped segments of their respective images.

Once all images are read by their overlapped sections, the clipped ECO, Water, QC and Cloud are positioned in their zeroed arrays based on the overlapping area location. Finally, this function returns the ECOSTRESS, QC, Cloud, and Parent Water mask in the same sized arrays and in their specific locations.

### def Clipping\_LST\_and\_Water\_Mask

Clipping\_LST\_and\_Water\_Mask requires the following variables to work

* **Line\_Number:** A number referring to a line’s labeled value. This value comes from an image that has been labeled through some sort of labeling function. This number is used to isolate the line within “Ref\_Img”
* **Ref\_Img:** This image will be known as the reference image and will have a line identified and isolated which will be used to clip the other images in the function. \*
* **Og\_Ref\_Img:** This image is used when “Ref\_Img” is a labeled image and not a binary image. If step value is 1, the Og\_Ref\_Img will be an empty 2x2 matrix. \*
* **Img\_to\_Clip:** This image will be known as “Image\_to\_Clip” and will simply be clipped based on the bounding box calculated from the “Reference\_Image”
* **Expand:** This variable can be customized, but by default it is set to 100. This 100 is based on ECOSTRESS’s 70-meter pixel size resulting in a 7 km error window.
* **Step:** The step option has two valid inputs: 0 and 1. Step 0 is used for clipping the large total shoreline to the ECOSTRESS LST image. Step 1 is used for the individual lines in the canny edge detected ECOSTRESS image to the water mask’s shorelines.

\* Note: Ref\_Img is the **labeled** **version** of Og\_Ref\_Img. This is implemented for step 0. In step 1 Og\_Ref\_Img is ignored with a filler matrix

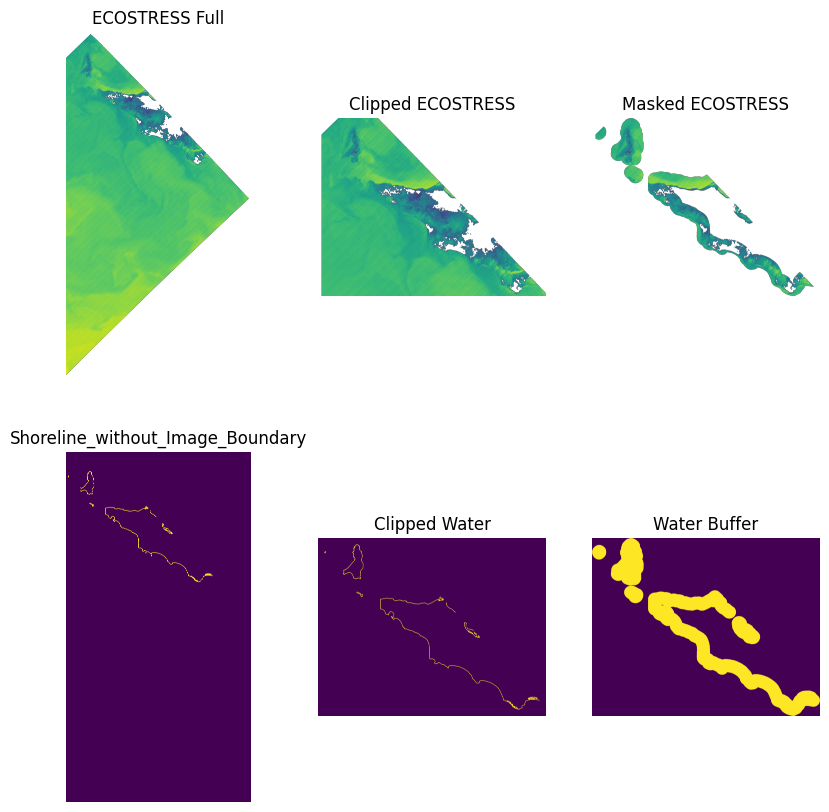
The “Clipping\_LST\_and\_Water\_Mask” function’s goal is to clip “Image\_to\_Clip” to the specific line’s bounding box in Img1 that was identified using “Line\_Number”.

The Clipping\_LST\_and\_Water\_Mask function first pads the “Reference\_Img”, “Image\_to\_Clip” and the “Og\_Ref\_Img” matrices to the “expand” variable on all sides. Expanding the image size is essential as moving the image along the image boundary will cause the image to loop around instead of stopping at the edge. The Reference\_Img is placed into skimage’s measure.region\_table to calculate the bounding box coordinates and their bounding box area. These areas are sorted by size and filtered by bounding box areas greater than 1000 pixels. This statement eliminates images which contain small shorelines that have a higher chance of finding an incorrect shift.

The “Line\_Number” variable is used mainly for step 1 as it helps isolate single lines and its water mask counterpart. Both steps create a mask keeping the desired shoreline(s) and removing the smaller shorelines due to the 1000 bounding box area limit.

For both steps, the bounding box area is split into Start and End values for both Column and Rows. These values are manipulated by the “expand” value to ensure the correct shift is within this expanded bounding box. This bounding box is now applied to all possible images and are returned to the user.

Example: ECOSTRESS LST was the “Img\_to\_Clip” and the waterWater Mask was the “Ref\_Img”. The Reference Image was reduced to its own adjusted bounding box. This adjusted bounding box was then applied to the ECOSTRESS LST image



### def Sorted\_Shorelines

Sorted\_Shorelines needs a single image to work:

* **Img:** Will be the ECOSTRESS edge detected image. This image is used to determine longest two lines.

The ECOSTRESS image is labeled using “measure.label”. After labeling, the image’s bounding box values and area are calculated using “measure.regionprops\_table”. The “length” of each isolated line is not provided with “measure.regionprops\_table”. In order to calculate the “length” of the line, the frequency of each label is calculated and used as the length of the line. This line length is then added to the table of values for all the lines within the image and is used to sorted the table by line length. This function returns the new “Sorted\_Table” and the “Labeled\_img” for further processing.

### def Endlines

The goal of Endlines is to identify the position of pixels that have one neighboring pixel within the eight pixels around the center pixel.

The EndLines function is applied to an ECOSTRESS image the contains a single line. This function first uses skimage’s “morphology.skeletonize” to reduce the amount of stray branches in the ECOSTRESS line. This function will apply the scipy.ndimage’s “binary\_hit\_or\_miss” function to detect pixels with only one connecting pixel in the surrounding 8 directions. Each “binary\_hit\_or\_miss” is summed into one result named “Combined\_Results”. The locations where there is a value of 1 is where the endpoints are located. The locations of all endpoints are zipped into a list. Circular lines will not work with this method so for these cases, the top left most pixel will be used. This function returns the skeleton image and the locations of all the eligible endpoints.

### def MatchingOverlay

Matching overlay requires 4 items:

* **Eco:** Isolated ECOSTRESS line. This step involves simplifying the line to eliminate stray branches.
* **Water:** Clipped water mask based on ECOSTRESS image
* **Expand:** 100-pixel buffer
* **Endpoints:** Location of all the lines endpoints. These endpoints could be from anywhere on the line where the line terminates.

The first step in this function is to extract the endpoint value. This endpoint usually is the top left most endpoint. This endpoint will have a buffer box with the endpoint at its center. This buffer box will be expanded by 100 pixels in the cardinal directions. The location of all shoreline pixels within this box are extracted and altered to shift values using the endpoint’s location and the expansion value. This option was used in order to use np.roll() in the Matching\_Shift.

### def Matching\_Shift

Matching\_Shift is the main step in calculating the best position for the ECOSTRESS edge detected lines. This function uses 3 inputs:

* **Shift\_Coordinates**: Shift values for the top left most endpoint to match with the water shoreline pixels
* **Eco:** The specific ECOSTRESS line that is the focus of matching
* **Water:** The clipped water mask with respect to the ECOSTRESS line expanded bounding box

The idea for this function is to add the shifted binary ECOSTRESS image and binary Water mask image to count how many “2’s” appear in the resulting matrix. The Shift\_Coordinates values are unpacked based on the looped value “i”. Using these values with np.roll(), the image is able to shift into a new positions. At every position, the summed count of each overlapping pixel is calculated and placed as the “Best\_Position”. This value remains the best position until a new calculated shift contains a higher overlap and will be replaced. When the “Best\_Position” is the initial position, the “Best\_Position” value is 0 and requires an if statement to specify the X and Y shifts.

### Shift\_in\_Progress

The Shift\_in\_Progress is the heart of the ECOSTRESS Geolocation process. This function in used in the multiprocessing line of code. This function requires 6 inputs:

* Eco\_File\_Temp: This variable represents the ECOSTRESS Scene filename. This file name will be used to derive the Water, QC, and Cloud Mask if possible.
* Folder: This variable is the file path for the ECOSTRESS scenes. This Folder will be joined to the Eco\_File\_Temp.
* New\_Folder: This variable is the file path of the new “Shifted\_” folder created before the main code function is used.
* Parent\_WM\_File: This variable is the filename for the single water mask that could be used as the “Main” reference for shift calculations
* Use\_Parent\_WaterMask: This is a flag that can be set to True or False based on the users preference on if they would like to use a single water mask or individualized water masks for each scene
* Use\_Cloud\_Mask: This flag can be set by the user and will state if the cloud mask should be applied to their scenes.

#### File Setup

After extracting the Day of the Year (doy) from the file name, the QC\_File\_Temp variable is created by replacing “\_LST\_” with “\_QC\_”. Both the Eco\_File\_Temp and QC\_File\_Temp are joined with the folders and later checked for existence within the folder by using “os.path.exists”. These two “checks” are compared to each other to investigate if the QC file is present. This step is done to save time by canceling the function early before conducting any shifting.

#### Water and Cloud File Identification

Due to ECOSTRESS Collection 2 having different water and cloud mask filenames for their Swath and Tiled versions, this section of code will check if the Eco\_File\_Temp is either Swath (“\_L2\_”) or Tiled (“\_L2T\_”) and create the files accordingly. Since the cloud mask is optional, there will be an if statement proving if the cloud mask file path is True. If the file name exists than the Cloud mask will be opened as an array.

Depending on if the Use\_Parent\_WaterMask is to True, the files will be opened using the Clipped\_Parent\_Water\_Mask function. This function will compare the LST, QC and Cloud masks to a specified water mask. Only the overlapped regions will be used to the rest of the calculation. This step was implemented to address some water masks having irregular shorelines. In future ECOSTRESS collections, the water masks will be replaced with a different source.

#### Shore Availability Check:

In the case of water masks being present but not portraying any shorelines, there will be an if statement checking if a zeros matrix is the same as the water mask. If this statement is false then the ECOSTRESS and QC files will be opened as an array with GDAL.

#### Determining Acceptable Edges from QC file

The QC’s purpose function in this code is to obtain the limits of the image. The acceptable QC are selected by identifying the values that are not equal to -99999, 65535 (Highest value in 16-bit), and 0. These values provide a mask that will have its edge erased by a few pixels using cv2.erode. \* This eroded segment doesn’t include the edges detected in the water mask.

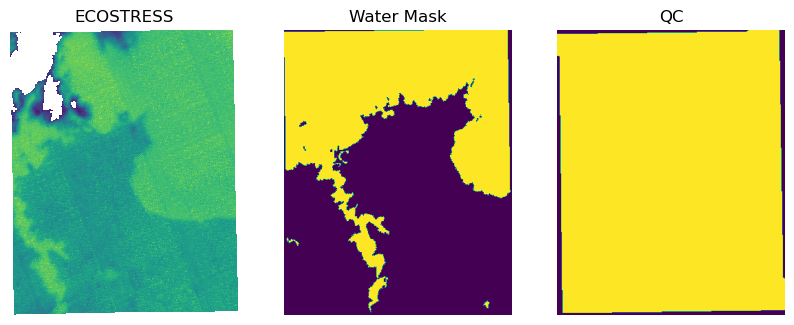
#### Removing Image Edges Detected in Water Mask

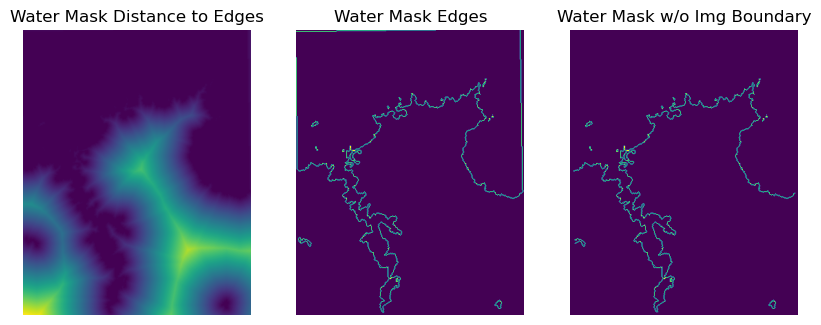
Like most satellite products, ECOSTRESS files are rotated and provide empty sections around the edges to fill up the array in python. These empty segments are filled with the value 255 which are converted to 0 to blend in with other 0 values in the file. This is important for the water mask as the land values are classified as 0 while water is 1. Setting 255 to 1 will cause slightly more issues with uniquely compiled areas from the AppEEARS ROI so setting this value to 0 works best for this case.

The Inverse\_Water variable is a switching the classification of land from 0 to 1 and vice versa for water. This matrix will need to be multiplied by 255 to allow cv2.distanceTranform to work. The cv2.distanceTransform calculated the distance one value is to another value. In this case, it will compute the distance 255 is to 0. Once calculated, we isolate the values equal to 1 resulting in our shoreline. This step is comparable to canny edge detection in terms of accuracy. Finally, the Eroded\_QC mask and the Water mask’s Shoreline are multiplied to obtain a matrix containing the actual shoreline. This new array is called Inverse\_Accetable\_Values\_Area.

Up to this point in the function, there are plenty large arrays that are no longer used. To remove these idle array’s, the “del” function is used with gc.collect(). Del allows for variables to be deleted. This step with gc.collect(), which looks for unused variables in the code, will confirm the deletion of unused variables going forward.

Note: Both the QC mask and the Water mask edges can be calculated with “measure.feature.canny()” however when running this file in batch with large swath tiles, this step takes nearly a minute per iteration. Using these unique steps instead of canny edge detection reduces the computation time.





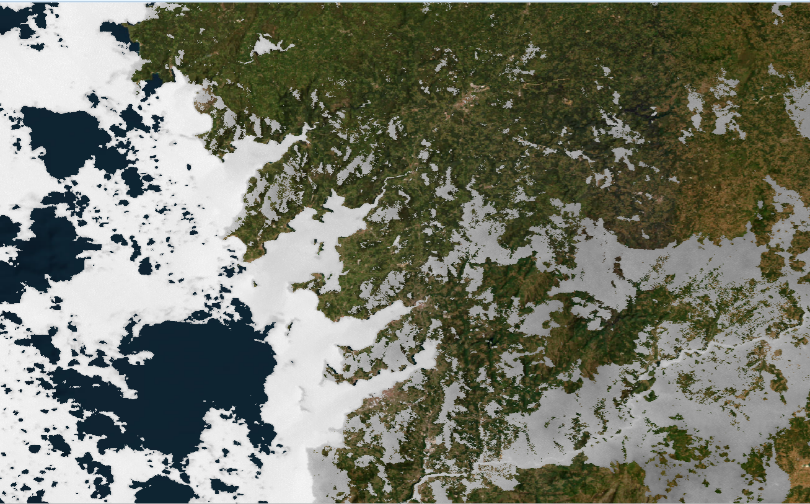
#### Applying Cloud Mask to LST and Water Mask

As stated in previous sections, the water mask is optional and will have a zero’s check to continue. This block starts with a conditional statement which checks to see if “CloudFile” exists. If the “CloudFile” exists, the code will compute the distance transform. This process is similar to the water mask however two masks will be produced.

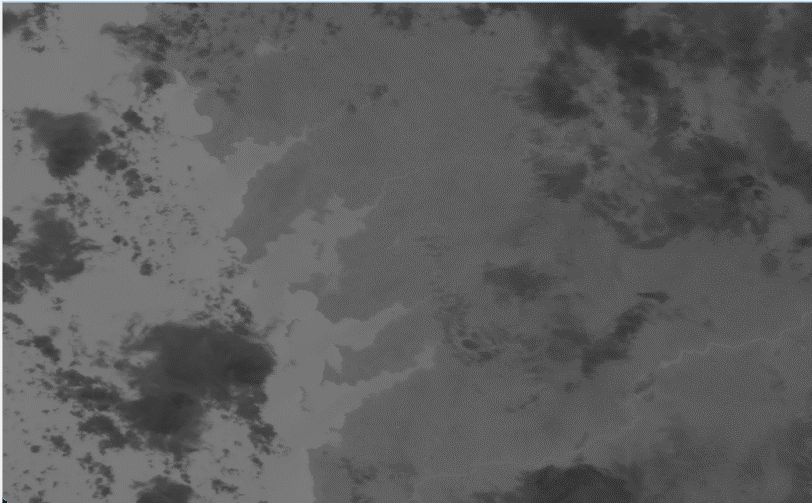
#### Cloud Mask on LST

**This step will be used if the Use\_Cloud\_Mask flag is set to True.**

In the LST canny edge detection, the edges of the clouds are found due to the clipped images. This is present in tiled products as this product has the cloud mask applied when downloaded. In some cases, the cloud mask outlines the actual land. The edges of the clouds could be beneficial but the noise within the cloud is not desirable. In this case a 5-pixel buffer is used as a threshold for the cloud distance transform. This 5-pixel mask is applied to the LST image



Spain Tiled: ECO\_L2T\_LSTE.002\_LST\_doy2023018190822\_aid0001



Spain Swath: ECO\_L2\_LSTE.002\_LST\_doy2023018190822\_aid0001

#### Cloud Mask on Water Mask

The cloud mask cannot be applied to the water mask directly due to the cloud mask being having the same geolocation as the LST file. To apply the cloud mask to the water mask, the cloud mask will have a radius of 100 pixels. Clipping the water mask to this 0-100 cloud mask removes shorelines that are completely covered by clouds while also allowing space for potential errors in the cloud mask’s geolocation.

Both the LST and Water Mask are masked by the Cloud mask in different ways based on the specific issues with the cloud mask. Regardless if the cloud mask is not existent, the unmasked LST and Water Mask are renamed to match the cloud mask altered arrays.

#### Identifying the Longest Shoreline

When calculated the edges form the water mask, there are cases in which the shoreline is not connected by a single pixel or two. To combat this issue the shoreline is dilated using “morphology.binary\_dilation”. This new dilated shoreline is used to find the largest bounding box that includes each shoreline. The Clip\_Water output is used to label the lines and find their area using cv2.connectedCompoenentsWithStats. The shorelines are sorted by area so the code can run the large segments first for order.

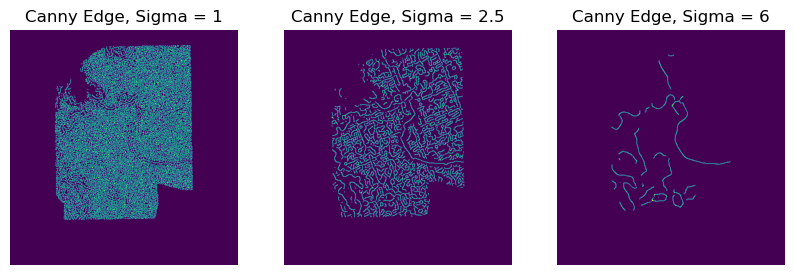
As seen after *Removing Image Edges Detected in Water Mask,* there will be a “del” and “gc.collect()” section for the variables that will not be used later in the code.

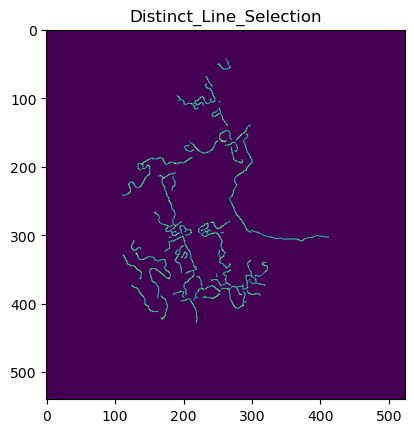
#### ECOSTRESS Edge Area Clip

The Clip\_Water image is expanded by 100 pixels in all directions representing the max shift error. This dilated water mask is applied to the LST image to avoid calculating unnecessary LST edges over empty water.

#### ECOSTRESS Edge Detection and Line Selections

Canny edge detection is applied to the masked ECOSTRESS LST image. A blur applied to the ECOSTRESS is needed to help reduce noise and help highlight areas with higher than normal temperature gradients. This blur effect is applied the same LST image but with a higher magnitude of 6 to further isolate the lines however there will be reduce accuracy in the higher order blur. The higher order blur is used to help isolate the lines in the 2.5 blurred LST image by labeling each line in LST, then multiplying the images together. The unique values in this multiplied matrix, excluding 0, are the label values kept in the final “Distinct\_Line\_Selection” matrix. Lastly in this section, the “del” and “gc.collect()” lines will be added to remove the last few variables and arrays that will not be used in shift detection.



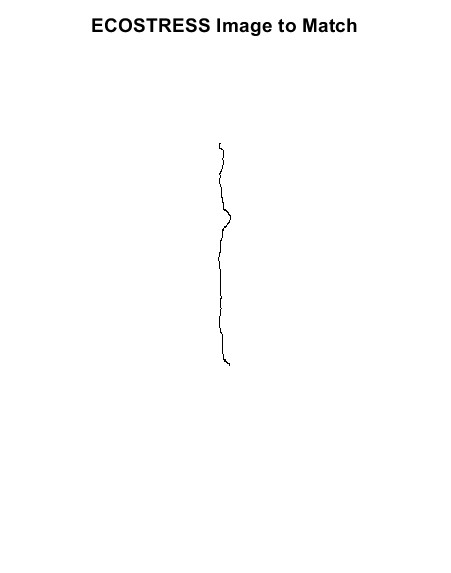
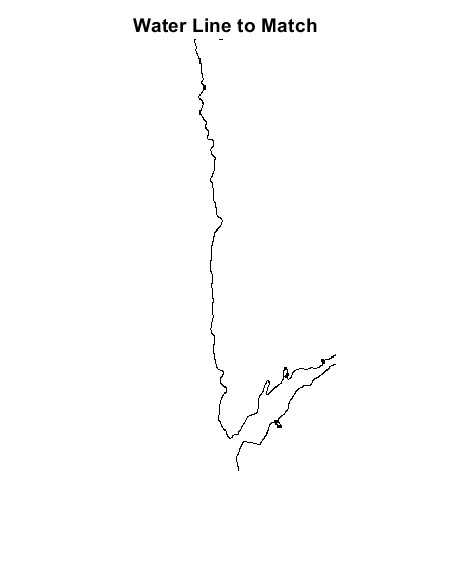


#### Selection of Top ECOSTRESS Edges

A shift is applied to every individual shoreline within the Water Mask. Each individual shoreline has its own bounding box and will clip the ECOSTRESS Edge image. Splitting up the shifts by shoreline and not the entire scenes is meant for faster calculations as large scenes tend to have unnecessary information. Lines identified in the LST image could all be short, however in some cases where the image is clear, a long shoreline can be identified. The long shoreline can be used to speed up the matching process as it can be the only line used to match the entire scene. The case for this step to go through is the difference between the top 2 lines identified in the entire image. If the longest line is greater than the second line by more than double it’s length as well has have a shoreline length of 50,000 pixels (Based on the sorted lines, the top number of lines are used to isolate the labeled ECOSTRESS image into a new labeled image.

#### Acquiring Eligible Shifts

This section of the loop uses the Clipping\_LST\_and\_Water\_Mask function to clip the isolated shorelines with the individual ECOSTRESS line. Once these two images are clipped to the new bounding box, the EndlLines of the ECOSTRESS image are calculated and used to avoid clustered lines. Lines are considered clustered if the line has more than 6 endpoints. If the line has less than 6 endpoints then the line is dilated by a radius of one. This dilation is used to account slight inconsistencies with water mask and ECOSTRESS edge lines due to temperatures. This new dilated line is implemented into the MatchingOverlay function. This function will find the best overlapping position between the ECOSTRESS line and the water mask by identifying the number of overlapped pixels within a maximum 100-pixel shift. This 100-pixel shift is shown as the blue box in the figure below. The returned values from this function are the Col\_Shift (x-axis), Row\_Shift (y-axis), and a ratio between overlapping pixels and the total number of pixels in the individual ECOSTRESS image. This ratio is used with Row and Column shifts to have some order in shift possibilities.



#### Shift Check for Combined Top Lines

Once each ECOSTRESS line is shifted for each clipped individual water mask, the temporary shifts are used to match the total individual water mask. These shifts are ordered by the ratio\*area value. The shifts are inserted, Matching\_Shift function to test the viability of each shift possibility. The best shift for the individual shoreline is stored added to the “Main\_Matching\_Stats”. The “Main\_Matching\_Stats” table will be used on the entire water mask and Distinct\_Lines\_Selection ECOSTRESS image.

#### Testing X and Y Shifts

Similar to the previous subsection. The best shift found from each individual water line will be used for the overall water mask shift test. The final shifts are returned from the Matching\_Shift function and may be used the following section “Applying the Shift”

#### Applying the Shift

This section is commented out to avoid any unwanted file creation without approval from the user. This section first creates a new subfolder within your current defined folder. It first checks to verify if this custom “Shifted\_Folder” exists before creating it. The LST image is opened once again with GDAL and creates a “New\_File” with the original LST name but with “Shifted\_” in front. The reason why the name alteration is in the front of the file name and not the end is due to visibility of file names in programs such as QGIS and ArcGIS due to their length. If the user decides to alter the naming convention they can do so at their own preference. Once this file is created, the file is opened with the “Shift\_File” variable name. This file obtains the set of coordinates and pixel resolution via “ .GetGeoTransform()”. The shifts found from running the code are multiplied by the resolution of the pixel. The shifts found are based on matrix values. Once updated to the pixel resolution, the shifts are added to the initial x and y starts. The new values are now added to the variable “gt\_update” which is applied to the “Shift\_File” file. Lastly all files are set to None to free up so memory.

## Main

### Main function

#### Reading and Creating Folders

The first step in the main code is to set the folder path. This folder path is your specified folder location. This folder path will be used to create a new sub-folder with prefix “Shifted\_”. Once this new folder is specified, the New\_Folder variable will be checked for its existence. This code will create shifted copies of your LST images. This sub-folder will be the destination for each possible scene.

#### Set Variables

This code has 2 logic variables representing the use of a cloud mask and the use of personalized water masks. Cloud masks are important for identifying land features however there are some cases where surface temperature are identified as clouds due to similar temperatures between cloud and land. By default the use of cloud masks is set to True. This will remove the core of the clouds but will keep the edges of the cloud as they may actually represent the shoreline.

The next variable is “Use\_Parent\_WaterMask” this variable will be based on the existence of the”Main\_WM\_Full\_Name” variable. This variable was included due to some inconsistencies with water mask shorelines. In some cases, the shoreline is clipped by some shape resulting in irregular shoreline patterns. The option to set “Use\_Parent\_WaterMask” to True will allow the following file “Main\_WM\_File” to be the reference image. This file should be set by the user. The file used must include your ROI and be large enough to include shoreline surrounding your ROI for better matching results. Each image in your folder will be clipped to the Main\_WM\_File’s bounds and will reference this image for matching results.

Note: Having Use\_Parent\_WaterMasks as True will only work for files that overlap with Main\_WM\_File. When this variable is True, then this code should be limited to one location. Having a folder with images around the world will be ignored besides images which overlap your defined main water mask.

### LST Flag Comparison

Note: There are two versions of LST Flag Comparison. AppEEARS has Geolocation flags for ECOSTRESS however it is only for swath products (as of February 19th, 2025). Alternatively, there is a .txt file that includes every Geolocation flag from the start of ECOSTRESS to November 29th, 2024 located in this codes zip file.

#### LST Flag Comparison: Option 1 (Flags.txt)

The Flags.txt file is a list of every geolocation flag from July 9th, 2018 to November 29th, 2024. This list is outdated and does not include the scenes after November 29th, 2024. Additionally, these flags are specific to the Collection 2 image and should not be relied on for Collection 1 as some Collection 1 images have been reprocessed with improved geolocation for Collection 2.

To start with the flag filtering, the Flags\_of\_Interest are defined as “Poor” and “Suspect”. These flags can be altered to include the “Good” and “Best” scenes, but “Good” and “Best” scenes already have well positioned geolocation. Using these flags of interest, the code will open the Flags.txt file and analyze if each line of text, contains any of these two flags. If the line does contain the line, the Date of the scene will be extracted. The date is located between the 26th and 34th characters. Using datetime.strptime, this date can be converted from YYYMMDD to YYYYDDD where DDD represents the day of the whole year (e.i July 21st , 2024 is 2024203). From the same line, the time is extracted from characters 35 to 41. The full doy can now be created by combined “doy” with the converted date and the extracted time. This doy will then be appended to the “All\_SP\_DOYS” list located above this block.

Once the DOYS are obtained from the flags.txt file, the User’s folder will be sifted through to gather every .tif file with “\_LST\_” in their file name. Once the file name is retrieved, the doy will be compared to the ALL\_SP\_DOYS” for any matches. If the match is successful, the file will be appended to LST\_filtered\_files.

#### LST Flag Comparison: Option 2 (AppEEARS Statistics .csv)

The ECOSTRESS team and LPDAAC have been addressing the Geoflags not migrating to the statistics.csv in AppEEARS’s supplementary data. As of March 13, 2025, only the swath product has the geoflags but this python batch code will have a block prepared to receive the csv files from AppEEARS when every file contains a flag in their csvs. For each .csv in the User’s folder, the csv will be opened and read using csv.reader. The first line will be used as headers and if “File Name” and “Geolocation Accuracy QA” are present, each row will be analyzed for the ‘Bad” or “Suspect” flags. If the flags are found, the filename will be appended int LST\_filtered\_files2.

#### Multi-Processing

The batch calculation uses concurrent.futures.ProcessPool.Executor. This module allows for the code to run in parallel and not one at a time. By default, the max\_workers will be set to None, this represents your laptops max number of logical processors. This can be inspected in the Window’s performance tab in Task Manager but can be determined with os.cpu\_count(). With concurrent.futures.ProcessPool.Executor as the executor, each file the Shift\_in\_Progress function, file, Folder, New\_Folder variables, Main\_water\_mask, and Setting Flags will be applied to executor.submit. Once executor.submit is complete, the result (future) is appended into Hold\_Futures for storage.

#### Writing the CSV file

Once the multiprocessing step is complete, each iteration will be added into CSV\_File list. This list will be used with csv.writer to create a new csv file with your own file name. By default the name of the .csv file is “ECOSTRESS\_Correction\_results.csv”.

## Running Main

For convenience, the code runs the code within Python’s if \_\_name\_\_ is ‘\_\_main\_\_’. Freeze\_support also is added to allow for multiprocessing do be run safely. Lastly a timer is set to show the total run time in seconds.