Name: Jacob Lin

Date: 8/30/2024

Advisor: Dr. Yu

Project: Exploring satellite image registration with AI

Exploring Satellite Image Pairing with Feature Extraction and Matching

Some quick introduction, my name is Jacob Lin, and I am a incoming senior University of Maryland, College Park (UMD) studying computer science and mathematics graduating in May 2025. I have a concentration in machine learning for computer science and pure mathematics in Math. In this summer, I had the wonderful opportunity to intern at the Earth System Science Interdisciplinary Center (ESSIC) at UMD and the partnership of National Oceanic and Atmospheric Administration (NOAA). I am working with Dr. Fangfang Yu who is an renowned NOAA scientist who works on the calibration and validation of satellite instruments. In this small paper, I will go through the process from start to finish of what we've done during this summer!

The GOES-16 is a satellite operated by National Aeronautics and Space Administration (NASA) and National Oceanic and Atmospheric Administration (NOAA) launched on November 19, 2016, though it is nearly a decade old, we still use the instruments attached to the satellite to this day. Two notable instruments are the Geostationary Lightning Mapper (we will be referring it as GLM) and the Advanced Baseline Imager (ABI), these instruments take photos of the earth periodically and sends it back to the researchers. The image generated by the ABI instrument is a higher resolution and larger size, the image generated by the GLM instrument is lower resolution and smaller size. The goal of this project is to identify the key points of these images using algorithms and/or machine learning and pair the correct images produced by the ABI and GLM instruments.

How would one find the key points of the following image to the right produced by the ABI? As humans, it is very easy to identify the islands, clouds, and oceans within this image. We can easily identify some key points within this image; for instance, the ends of the islands and the small clouds. How would you implement the methodology of identifying these key points to a computer? One crucial feature we can easily identify is the difference in color between the cloud, land, and water. The clouds are white, the land is gray, and the water is black. Another prominent feature is the shape of the clouds and land. To enable computers to recognize these features, we can use the Scale-Invariant

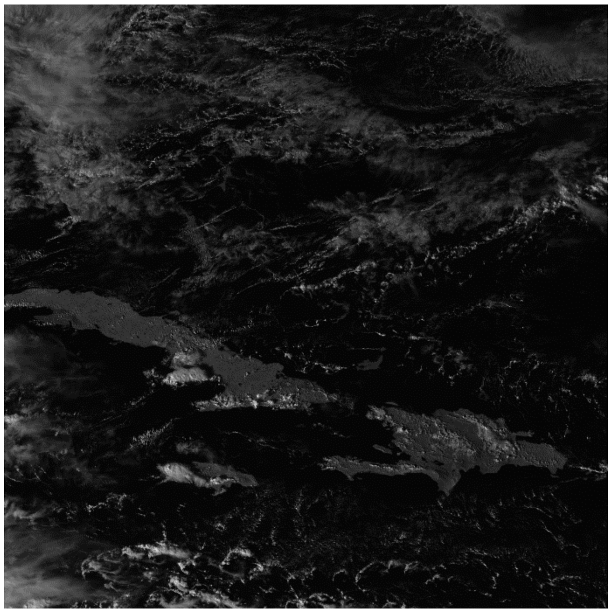
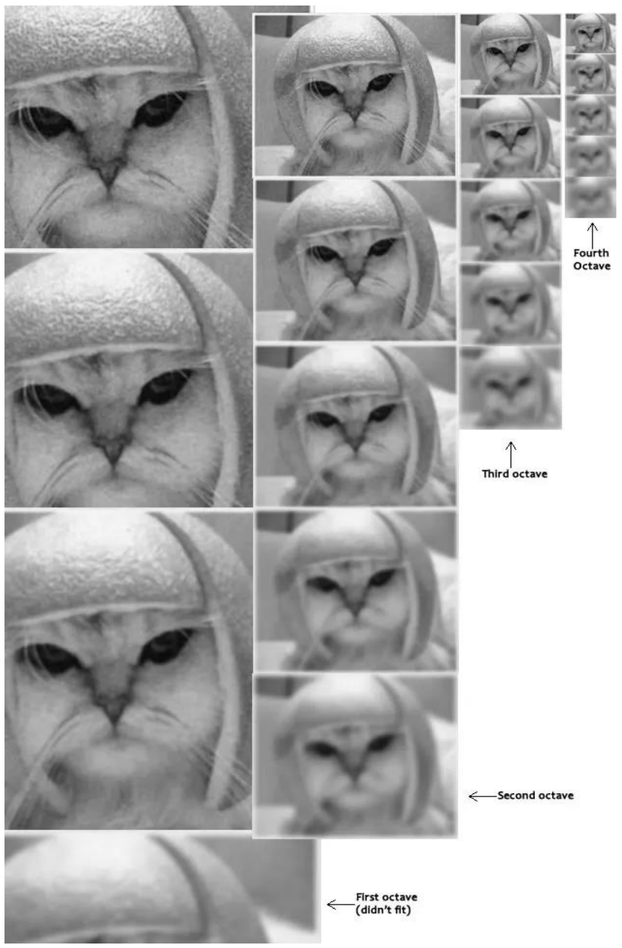


Image generated by the ABI instrument but shrinked to fit the page.

Feature Transform (SIFT) algorithm. SIFT is a powerful method for detecting and describing local features in images. It identifies key points by analyzing the image's scale-space and looking for points that are stable across different scales. This means that the key points identified by SIFT are robust to changes in scale, rotation, and illumination, making them particularly useful for satellite imagery where lighting conditions and perspectives can vary.

There are 3 main key steps of the SIFT algorithm. These steps are Scale-Space Construction, Difference of Gaussians, Key Point Localization. These steps are crucial to algorithmically generate the key points of the image.

1. The first step is Scale-Space Construction, it is done by the following steps:



a) The scale-space representation consists of multiple octaves, each containing a series of blurred images. The first octave starts with the original image. To create subsequent octaves, the image from the previous octave is down sampled by a factor of 2 (i.e., halved in size). This down sampling reduces the image resolution, effectively shrinking the image dimensions by half in each step.

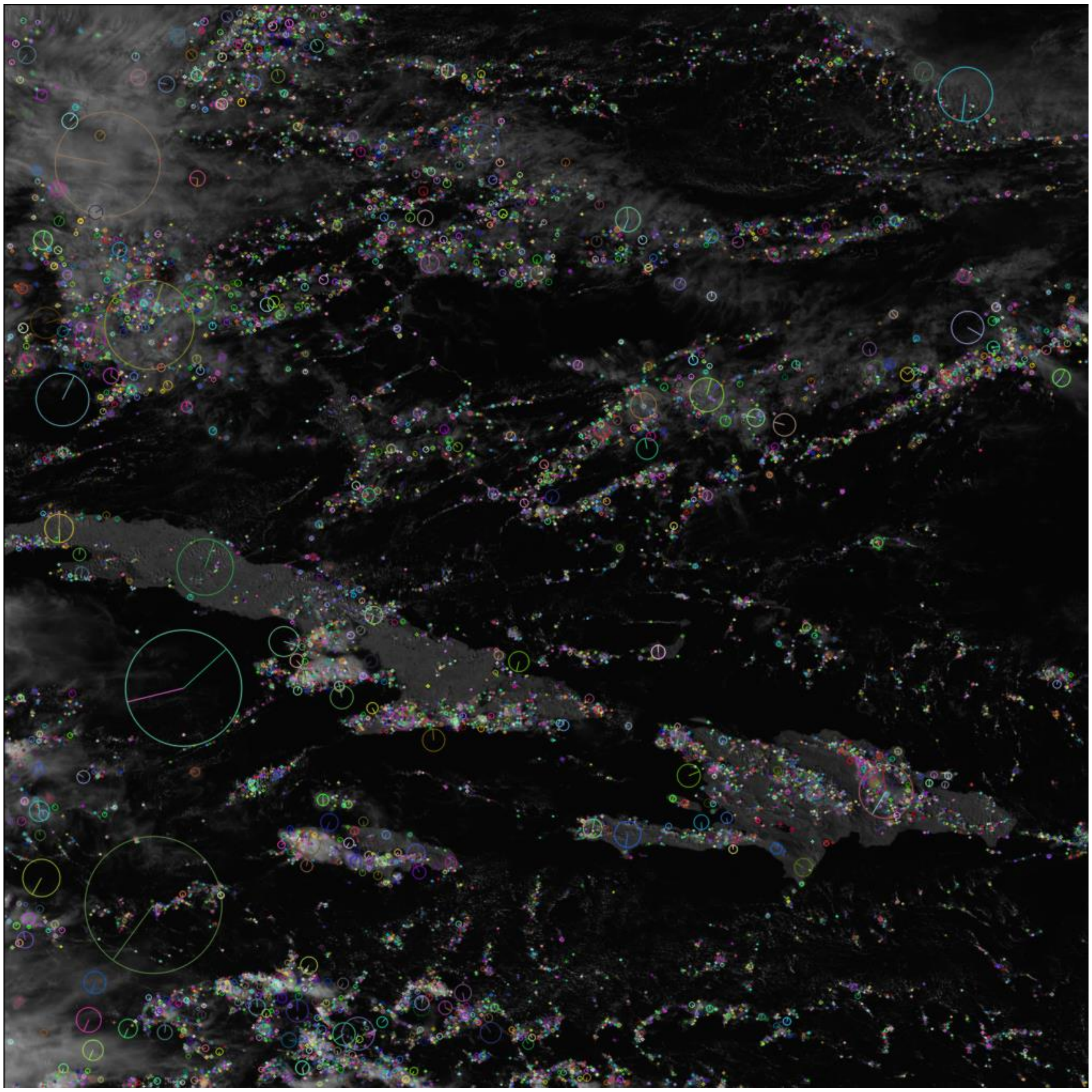
b) The process begins by applying a Gaussian filter to the original image at different scales. The Gaussian filter smooths the image by blurring it, effectively reducing the high-frequency noise and highlighting prominent features.

2. Within each octave, the algorithm computes the Difference of Gaussian (DoG) by subtracting adjacent Gaussian-blurred images. This process approximates the Laplacian of Gaussian (LoG), which is a key point detection method that highlights regions of rapid intensity change (such as edges). The DoG images serve as a basis for detecting stable key points across different scales, as they emphasize areas in the image where the intensity changes significantly.

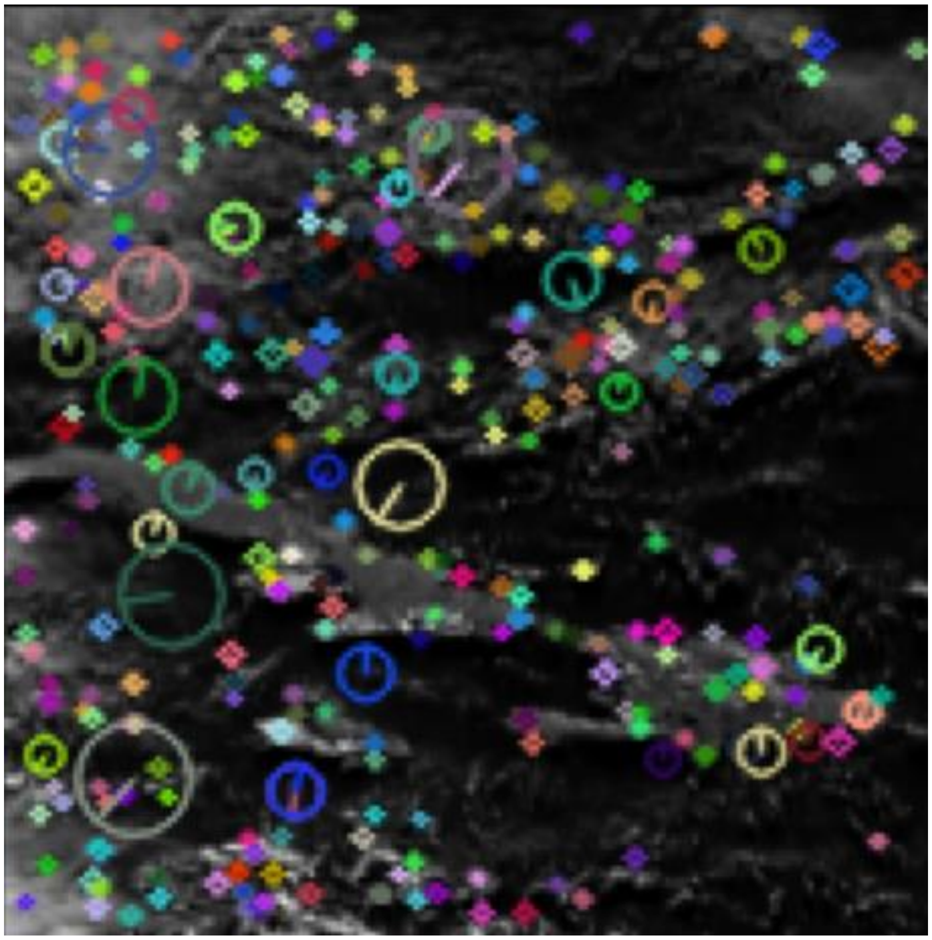
3. The DoG images, created during Scale-Space Construction, highlight regions in the image where intensity changes significantly. These regions are potential key points. To locate these key points, the algorithm searches for extrema (both maxima and minima) in the DoG images. This is done by comparing each pixel in the DoG image with its 26 neighbors (8 in the current image, 9 in the scale above, and 9 in the scale below).

We can see the examples of this in the diagram above. Observe how the second octave progressively becomes blurrier, and we can see where some of the important features are in the original photos like the eyes, mouth, and nose.

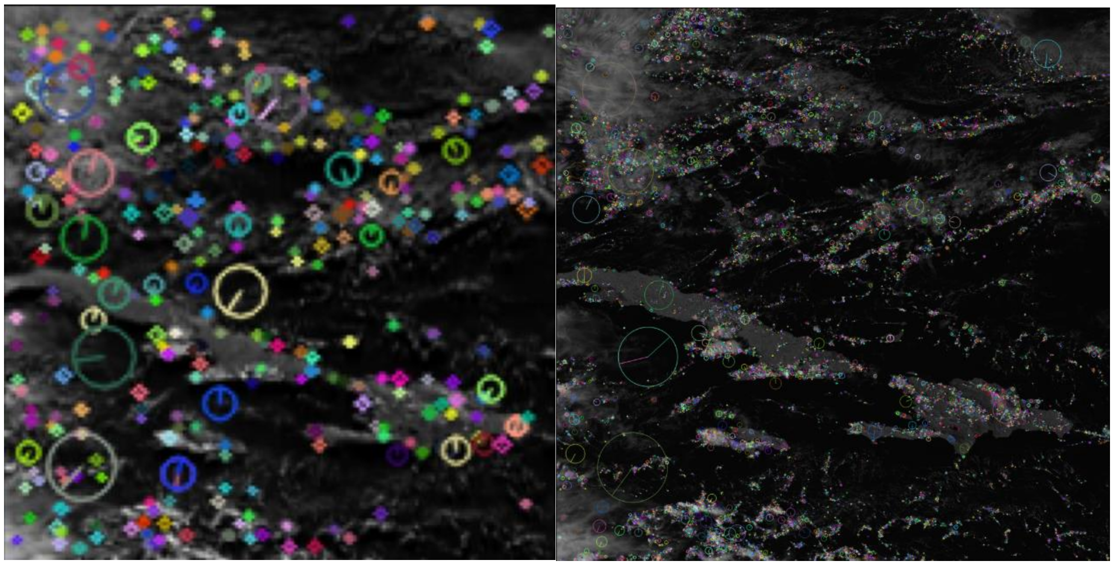
In this project, we used an open-source computer vision library called OpenCV in python developed by Intel. In this library, contributors to the project already implemented the SIFT algorithm which I used on the images produced by the ABI and GLM. Here is the result of SIFT algorithm to the image we have showed as an example of how to generate key points of an image.



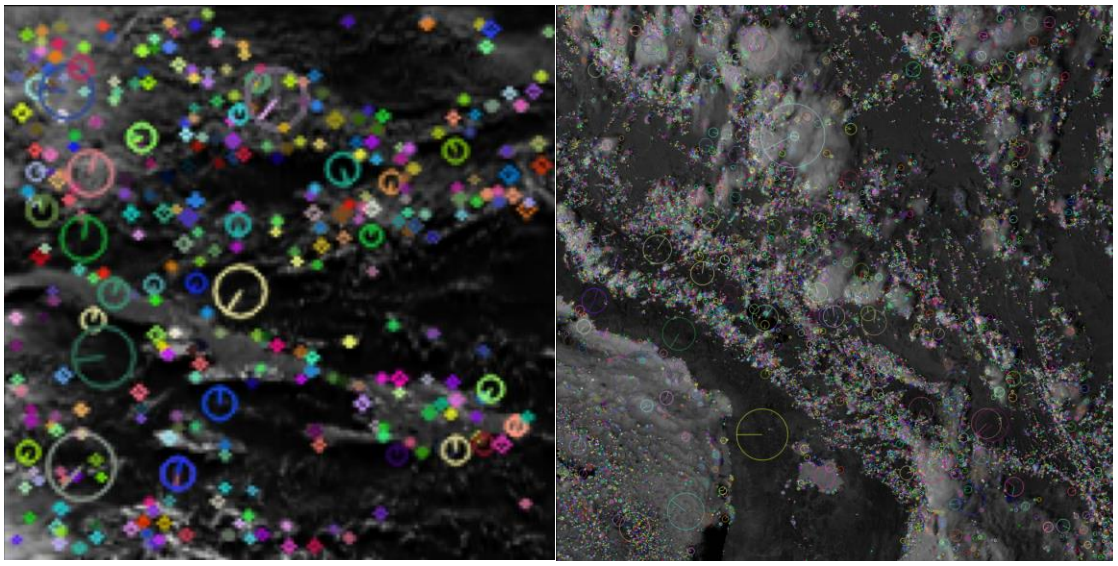
As we can see, each dot/circle is a important key point generated by the SIFT algorithm. Notice how the algorithm emphasizes the key points of the image on the clouds and the islands in the picture exactly how a human would identify these points. The larger circles show a large radius of importance. Here is the key points on the same image but generated by the GLM instrument.


We can see the low resolution of the image produced by the GLM instrument here.

Now the question is, given multiple images produced by ABI and GLM instruments, how can identify which images produced by GLM matches with the image produced by ABI. Firstly, we will put these images side by side and we can see some clear observations.



It is clear that some of the key points are very similar between these 2 images. Some examples we can see if the large circle on the large cloud in the top left of both images, in the GLM image it is blue, in the ABI image it is peach colored. Also looking near the bottom left, we have 2 larger circles which is similar between these, and lastly on the clouds, they are similar as well. Take a look at another ABI image which is different from the one above. We will also put this side by side.



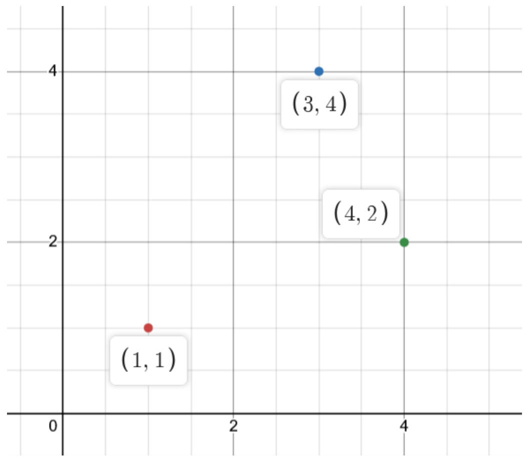
We can obviously see the difference between these 2 images both pictorially and with the key points generated by SIFT on these 2 images. Now the idea of solving our question of how to match the corresponding GLM image to the image produced by the ABI instrument is clear. We will do this algorithmically as the following with the following logic:

1. For every GLM and ABI image that we have, generate the key points for all of the images.
2. Take a single ABI image with its key points generated by SIFT, and brute force match every single point with the points in the GLM image.
3. The most matches between the ABI image and the GLM image will be the corresponding pair.
4. Do this with every single ABI image.

Now the question comes on how we can calculate which key points correspond to which point in the other image. One important fact that is deliberately left out for the sake of complexity is the information that the key points hold. While it looks like just circles and dots to us, there is more under the hood than meets the eye. SIFT in fact has 2 extra steps which hold the information that these key points have. The first thing that one can notice within the image with the key points is that the large circles have a line from the middle of the circle to the edge. That is the Orientational Assignment step in SIFT which finds the orientational invariance of the key point by computing the gradient of each pixel then creating an orientational histogram with 36 points (which is to represent the 360 degrees of the circles) then taking the largest invariance. The last true step of SIFT is the most critical, it is the key point descriptor. The key point descriptor, as the name applies, describes the key point. One might ask how it might describe the key point? Does it give it a funny name? Does it say what it is? The matter of fact is that the key point descriptor in SIFT is a highly detailed and mathematical representation of the local image region around a key point. Rather than providing a name or a direct description, the key point descriptor captures the distinctive patterns of intensity gradients within the region. This is done through a series of precise numerical values that encode the local structure and orientation of the image at the key point.

In this section, I will go through the specifics of the key point descriptors, feel free to skip to the next section. The key point descriptor is created by first computing the gradient magnitudes and orientations in a local neighborhood around the key point as we have described in the Orientational Assignment step. This area is divided into a grid of smaller regions, typically a 4x4 grid. For each sub-region, an orientation histogram is created by accumulating the gradients' contributions according to their direction, weighted by the gradient magnitude and a Gaussian window centered on the key point. Each histogram usually contains 8 bins, corresponding to 8 different orientation directions. The final descriptor is a concatenation of these histograms, resulting in a feature vector with 128 elements (4x4x8). This vector effectively captures the local appearance around the key point, making it unique and distinctive. These descriptors are not arbitrary numbers but carefully calculated values that ensure invariance to common image transformations, such as scaling, rotation, and changes in illumination.

To summarize what the key point descriptor is for those who skipped the previous section, the descriptor is a list of numbers stored in a special way called a vector. Each number is distinct since not all key points are the same. The next natural question to ask is what the importance of these vectors are. As said in the previous sections on brute force matching key points, we want to take every key point whatever pair of images that we want to try and match and match the key points. We do this by finding matches based on the metric distance, to be more precise, the Euclidean distance. Take this very simple graphical example. Let's say our vector to be a 2x1 vector which we can easily plot out in a graph. Given the image to our left, assume that the red vector is an ABI vector, and the blue and green vectors are all the vectors in the GLM image. We want to find whatever the closest point is to the red vector since that will be the closest to that match. Remembering what high school math tells us, we can easily find the distance between these points by the distance formula inspired from the Pythagorean Theorem which gives the following.



$$\text{Distance} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_1)^2}.$$

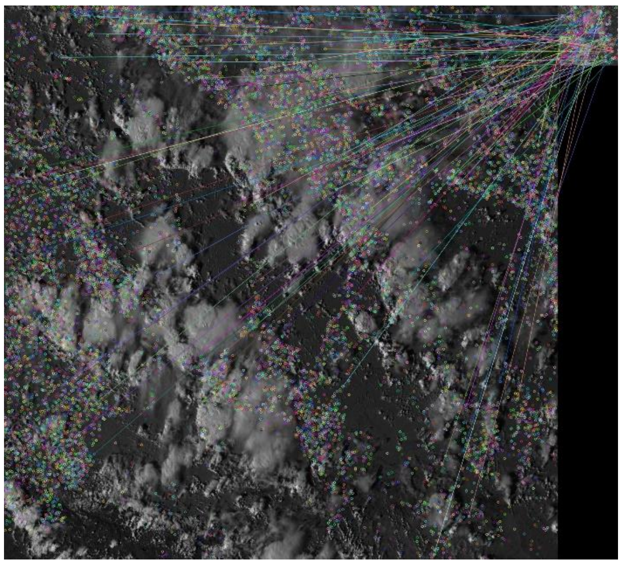
Doing the match yields the following. The distance from the red point to the blue is 3.605 and the distance from the red point to the green point is 3.162 which means that the green point is the closest point to the red point which means that the red descriptor is the closest to the green descriptor. Now the vectors we are dealing with in SIFT are 4x4x8, one might ask how we do this with that kind of vector. (I'd advise readers to skip to the next page with the formula) We will take a chapter out of our advanced calculus textbook. First, we want to flatten the 4x4x8 vector to a 128x1 vector. Then we can apply the Euclidean distance formula for 128×1 vectors also known as vectors in $\mathbb{R}^{128}$ meaning in the 128th dimension. Given 2 of these vectors A and B, we will index each of the entries of the vectors by i.

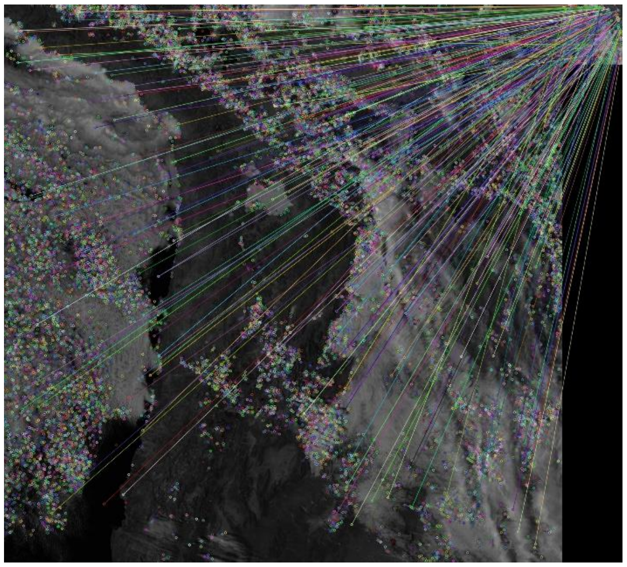
Thus, we can have the distance formula as the following:

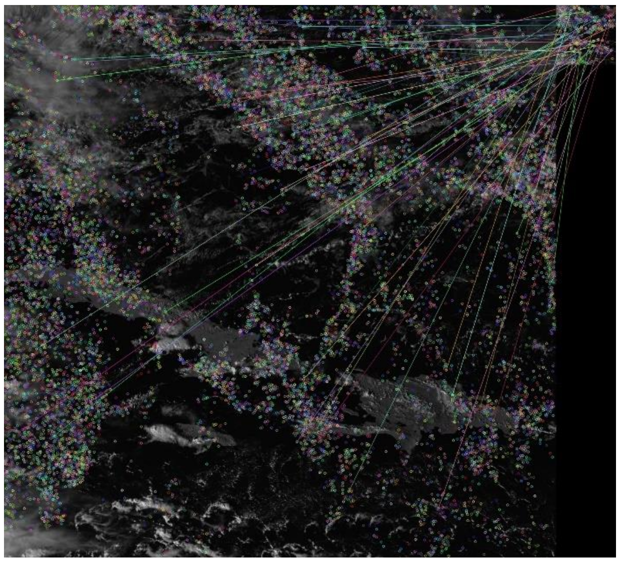
$$\text{distance} = \sqrt{\sum_{i=1}^{n}(A_i - B_i)^2} = \sqrt{(A_1 - B_1)^2 + (A_2 - B_2)^2 + \cdots + (A_{128} - B_{128})^2}$$

which is just a fancy way to steroid the distance formula from high school math.

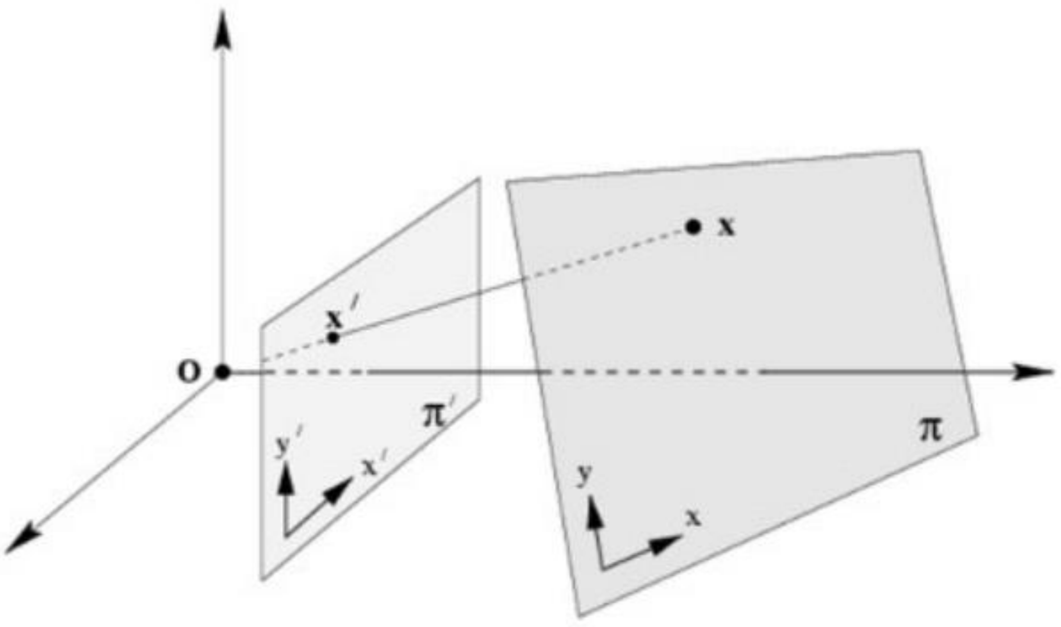
So finally with everything being said, that is the technical explanation of how we matched the images algorithmically. With this method, we have the ability to match all of the possible image pairs one can get. Here is the result of the SIFT key point generation and the brute force matching.







One might think we have done an amazing job at this assignment, and we completed this first try. And we thought so as well, so me and Dr. Yu decided to assess these key point matches. The natural question which one might ask now is how would you assess this image? One cool thing from OpenCV is that you can get the coordinates of these matched points, for example you can get the following ((1024, 20), (24, 54)) the first tuple (1024, 20) is the matched key point coordinate in the ABI image, and the second tuple (24, 54) is the matched key point coordinate in the GLM image. This is important in the method we used to assess the accuracy of the matching done by brute forcing the matching of the key point descriptors. While there are multiple ways to do the assessment, the method we used is what we called the Homography Matrix assessment. We use a homography matrix because OpenCV already has an implementation on this again and we don't need to make our own algorithm to save time. A homography matrix described by OpenCV is "Briefly, the planar homography relates the transformation between two planes (up to a scale factor)". A good visual representation is the following:



Where we have 2 planes, and it describes the relationship between the two of them. We can formulate this transformation by the following equation:

$$s\begin{bmatrix}x'\\y'\\1\end{bmatrix}=\boldsymbol{H}\begin{bmatrix}x\\y\\1\end{bmatrix}$$

Which **H** is a 3×3 matrix, thus we have

$$s\begin{bmatrix}x'\\y'\\1\end{bmatrix}=\boldsymbol{H}\begin{bmatrix}x\\y\\1\end{bmatrix}=\begin{bmatrix}h_1&h_2&h_3\\h_4&h_5&h_6\\h_7&h_8&h_9\end{bmatrix}\begin{bmatrix}x\\y\\1\end{bmatrix}$$

Where $h_1, h_2, \dots, h_9$ is are the entries in the **H** matrix. So first we will define what a source point and what a target point is. A source point, as the name suggests, is the point in the original image that we want to transform to another image. A target point is the transformed point from where you want the source point to match to. Let us also introduce the idea of an error, like in real life, pretend you predict the score of your favorite football team to score 30 points, but in reality they have scored 28 and you lost a bet against your friends. Although it is sad that you lost a bet against your friends, but what is here to notice is that the error between your guess and the actual score is 2 points since you are off by 2. So, we can formulaically define the error to be

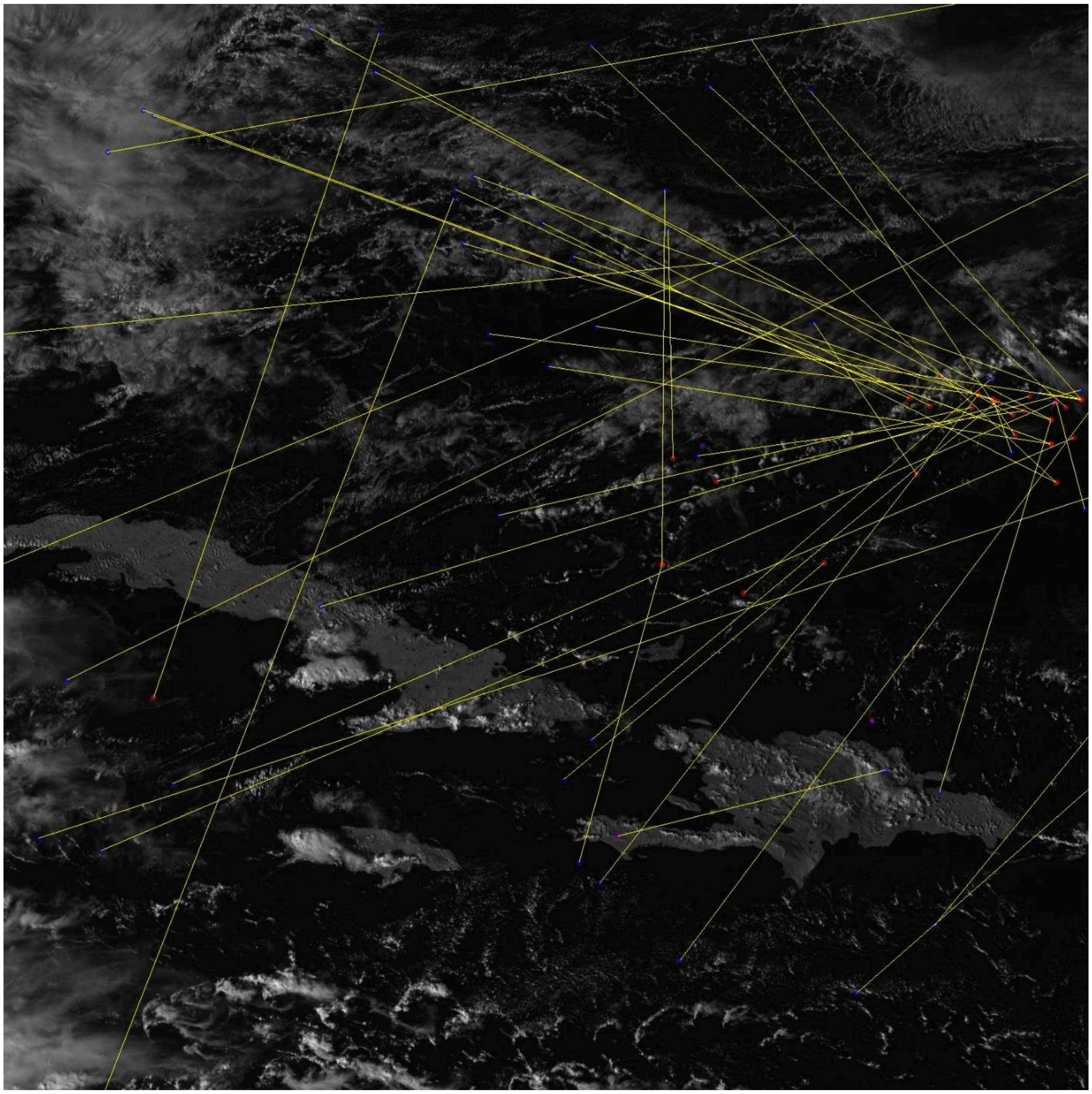
$$\text{Error}=|\text{Predicted - Actual}|$$

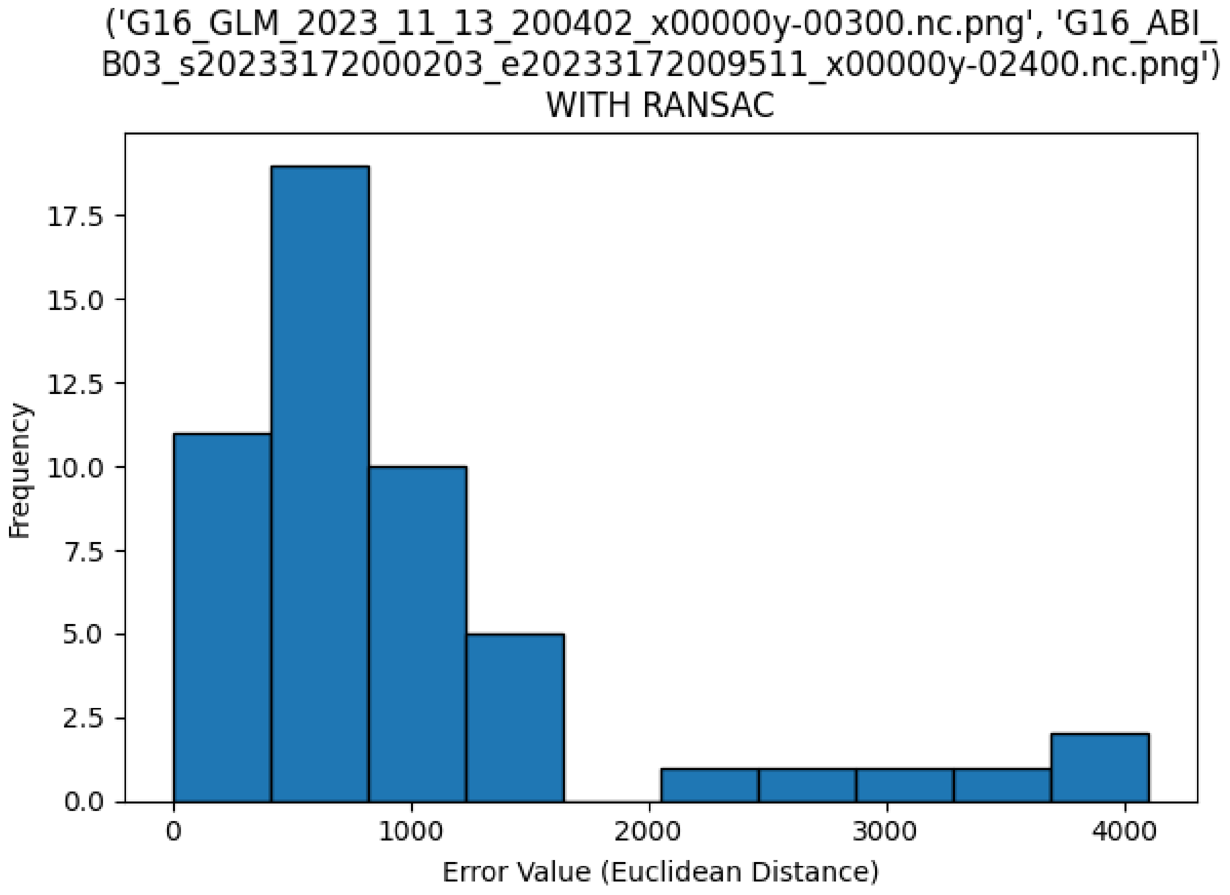
Which doesn't matter the order of predicted and actual, it gives the same idea. The general idea of this assessment is given a list of source points (which in this case we will use the GLM image), then we will generate a homography matrix which will minimize the errors between all of the points.

Having said that, how do we implement that on what we have? Once again, we rely on the implementation of OpenCV's homography matrix method, we can input all of the source points and target points and OpenCV will return us a homography matrix which will minimize the errors. We can predict the points by doing the following:

$$\boldsymbol{H}\begin{bmatrix} x_{source} \\ y_{source} \\ 1 \end{bmatrix} = \begin{bmatrix} x_{predicted} \\ y_{predicted} \\ 1 \end{bmatrix}$$

Then we can get the error by using the Euclidean distance as described before. Let us try to assess the image that we've used throughout the paper. The following will also show the relative histogram for the assessment as well.



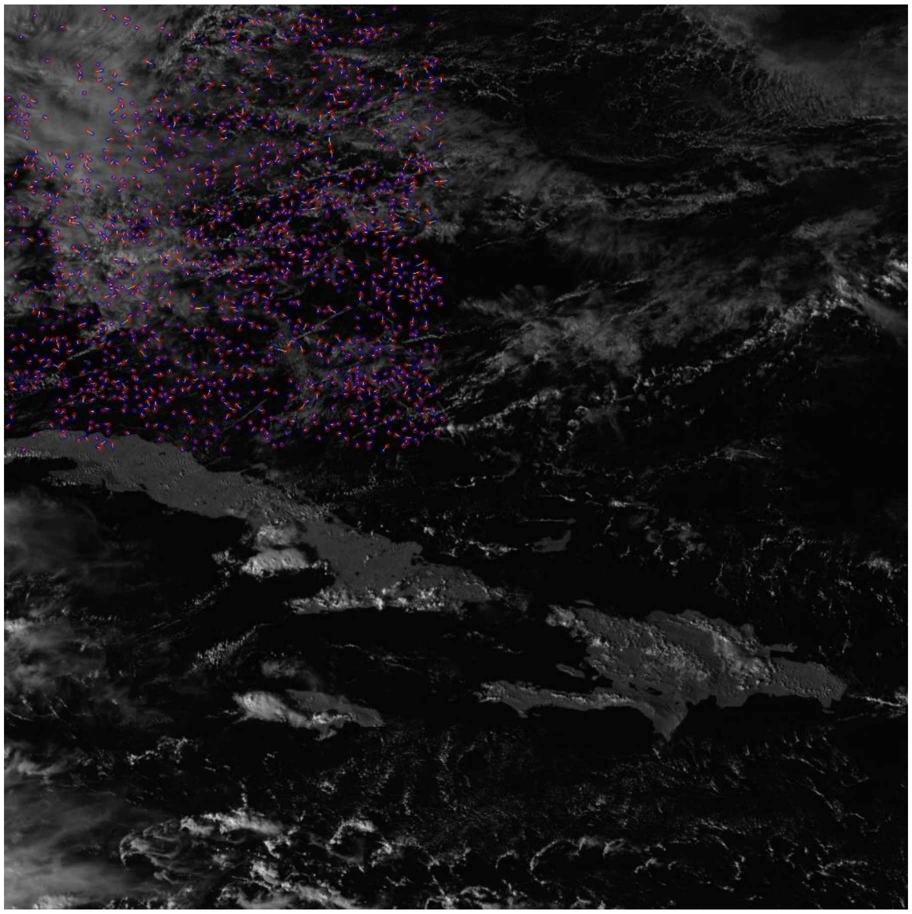


As we can see, this assessment was terrible. The majority of the errors are about 500-600 pixels away from the actual point which is insanely inaccurate. You can also see in the map picture that the yellow line between the red and blue points was extremely large which also sign of inaccuracy.

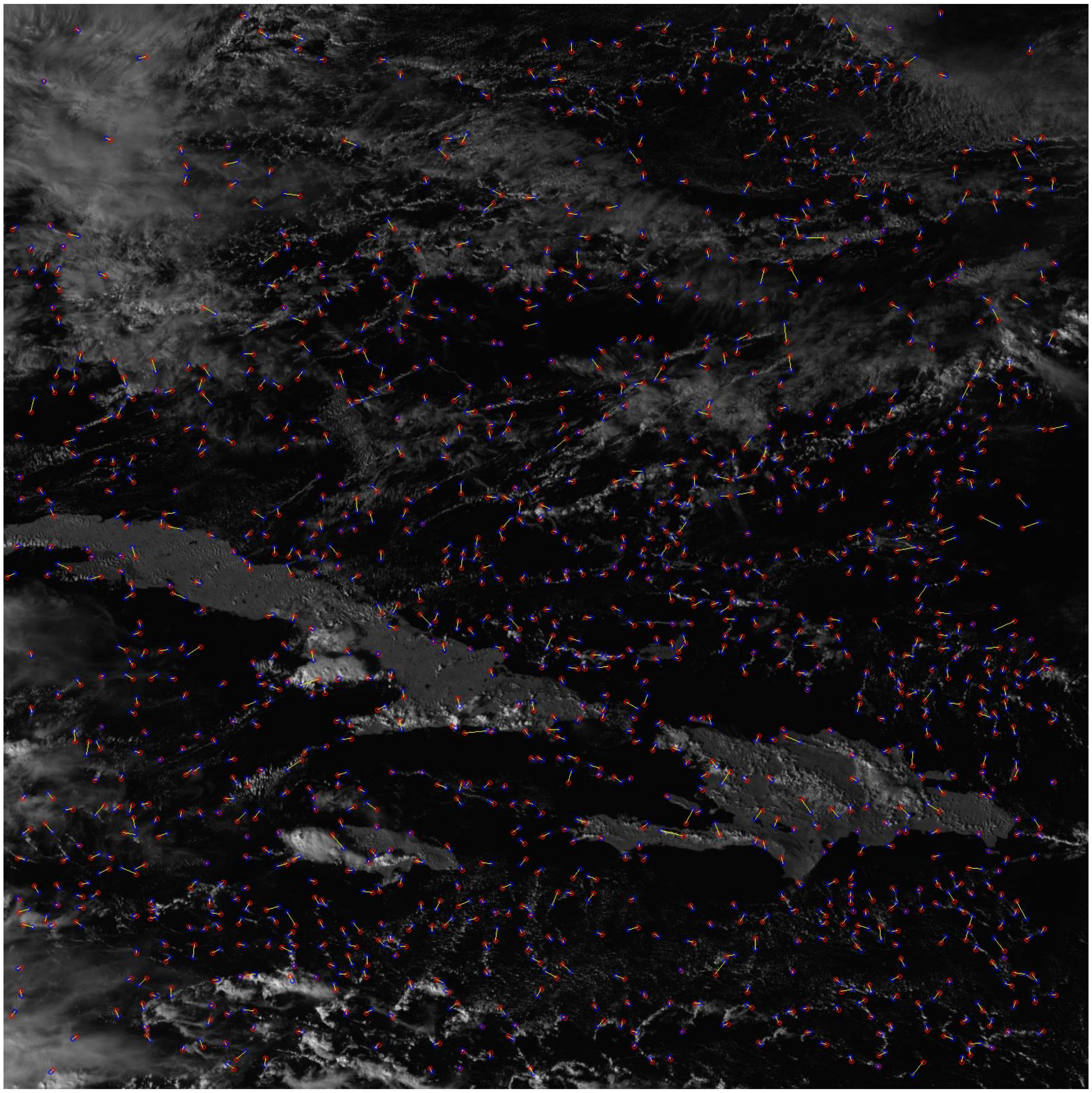
After this we realized something was wrong. Dr. Yu pointed out that the matching was incorrect. Since I had limited knowledge about how the brute force algorithm from OpenCV works, I had to resort to another solution. One thing I found with Dr. Yu's advice of using pretrained machine learning models is Magic Leap's pretrained machine learning model. This model is named SuperGlue. The usage of SuperGlue is very niche, what it does is takes the key points of an image generated by SuperPoint (another pretrained model which I will go over) or SIFT and use the power of neural networks to pair these key points up. I played around with SuperGlue and its limited documentation, and the results were disappointed. With limited machine learning knowledge since I have not taken a course on Machine Learning yet, I had troubles trying to get the tensors in shape to match what SuperGlue needed. SuperGlue also had a limitation where it can only accept 256 points and we had some images with more than that many. MagicLeap did have another model which uses SIFT directly, but according to MagicLeap "We do not intend to release the SIFT-based or homography SuperGlue models" which really sucks, so with limited time left, we decided to switch to using SuperPoint.

SuperPoint and SuperGlue goes like bread and butter, MagicLeap's intention was to use SuperPoint and SuperGlue coincidingly. After realizing this, we decided to make the switch. As said before, SuperPoint is a pretrained machine learning model which is intended to identify the key points within an image. Then the idea is to directly feed the output of SuperPoint into SuperGlue to match the key points. I wish I can tell you how SuperPoint and SuperGlue works, but my limited knowledge hinders my ability to do so.

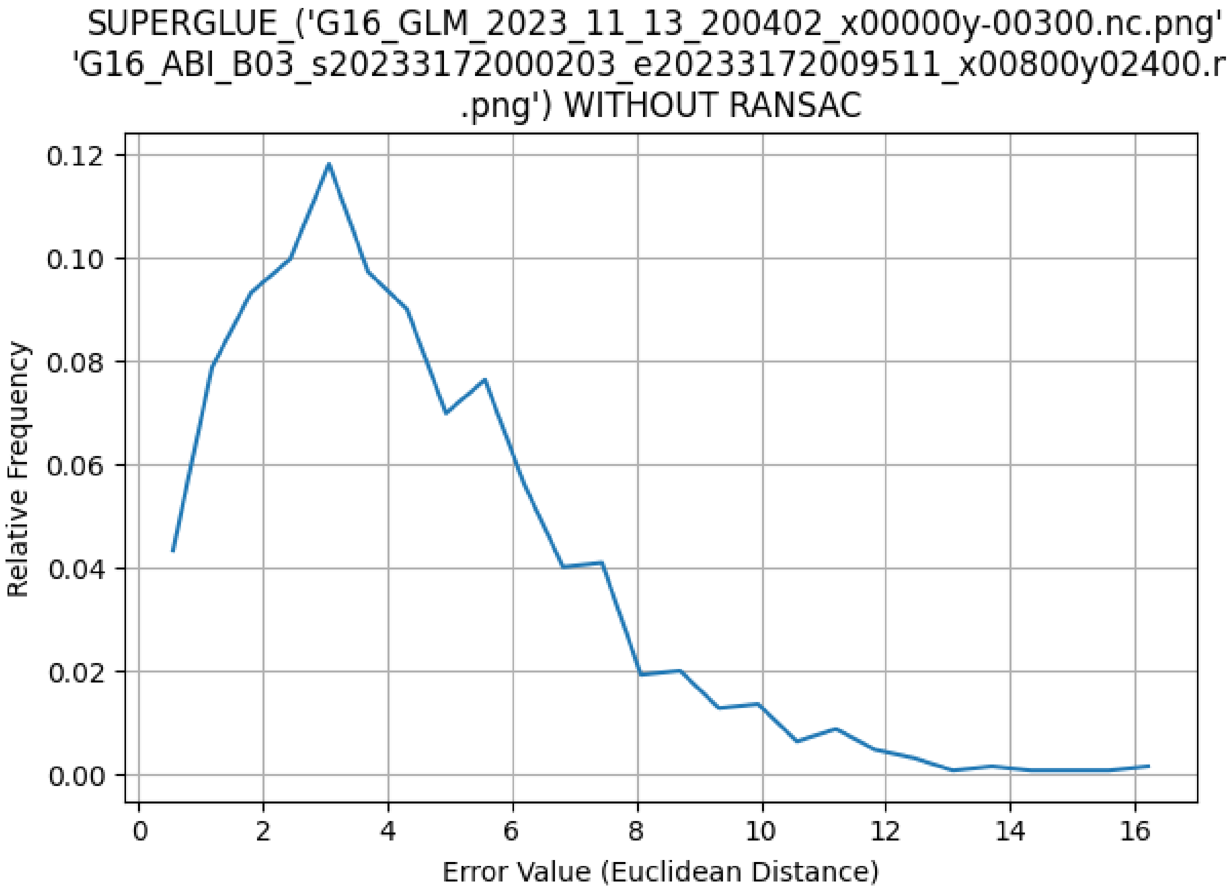
Trying to implement these pretrained models was not smooth sailing as well. Thankfully MagicLeap provided an example of using SuperPoint and SuperGlue simultaneously which helped tremendously within the implementation. One blocker that we faced is trying not to upscale or downscale any of the images and keeping the image as original as can be, but this proved to be impossible. This is because when an image is small and low resolution, SuperPoint and SuperGlue can be super inaccurate and may not generate any key points or make any matches in general, this is mainly referring to keeping the GLM image clean. The other side of the coin is an issue as well, when an image is very large and high resolution, this is possible with a strong enough computational power, but with my very limited laptop with a somewhat high end graphics card to process (a Nividia 3060 GPU), it still cannot handle the sheer size of the pretrained model and often errors by not allocating enough memory to do a task. So we had to meet in the middle. I ultimately decided to shrink the ABI images by half and upscale the GLM image to the same size. SuperGlue and SuperPoint works the best when both of the images are the same size. After implementing what I just ranted, we get the following results below. My first initial reaction was like pure relief since it is



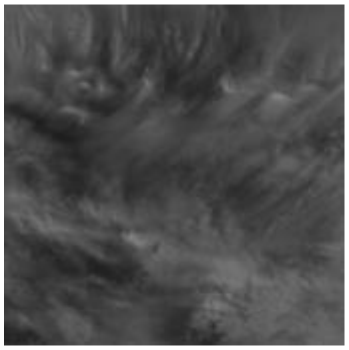
exactly what we wanted with minimal looking errors since the yellow lines are very small. But to you it looks wrong, like why is it all in the first quarter quadrant of the photo? Well remember the process on how I got it to run on my machine? I had to first half the image then run SuperPoint on it, thus all of our target points are halved as well. So let us multiply all the vectors by 2 to get a true representation of what the photo looks like. I will also provide the relative histogram that is produced.



Here is the image, we can see how beautifully this worked. Notice the edges of the island being key points, the edges of the clouds, the pretrained model is truly a masterpiece. Notice how miniscule the yellow lines are, some not even visible to the naked eye. And observing the histogram below, we see a majority of the errors are only 5 pixels apart!



Remember the issues of not trying to modify the original image as much as possible? After getting satisfactory results, we decided to revisit this problem to see if there is any potential ideas to solve it. One idea was to take subsets of the ABI image that are the same sized images as the GLM. For example we have the following image below which is a 170x170 subset of an ABI image.



This still brings the problem we brought up earlier, where the smaller images are more inaccurate and may not produce any results which is the case. Another idea was just to take the center of the ABI image which is half the size, while this is a good idea, we still ran into the problem of the smaller GLM images not producing any key points to compare to. So that's where our project portion ends. We would love to try and keep the original images as pure as possible, but with current capabilities, it does not seem like its viable.

That concludes our part of the summer intern project, me and Dr. Yu will be going into uncharted territory and trying to make our own machine learning models. Where the project currently stands, I am satisfied with the results. I enjoyed this project a lot and working with Dr. Yu was amazing as she is very knowledgeable within this field. She always has ideas on how to resolve problems that I am having and helped me guide my way through this internship. I have truly learned a lot about computer vision and artificial intelligence in this summer and I hope there is more to come!