



Towards Finding Contrails in Cloud Images

—

Director: Professor John Barron

Tuochaolong Zhang

Western University

2016 May 24th

Computer Science 3380Z



Introduction

This Project topic is original posted by NASA Space Apps Challenges. Here is the introduction on this project page (spaceappschallenge.org) .

“On clear or partly sunny days, people might look up at the sky and see straight lines of what appear to be clouds or white smoke. These lines are not smoke or natural clouds; they are contrails produced by aircraft. Contrails form because water vapor from jet engine exhaust passes through a cold and humid part of the air at high altitudes. Sometimes the jet that created the contrails is not visible overhead because winds aloft have blown the vapor trail into the observed area after the jet has passed. Naturally occurring high thin cirrus clouds do not form straight lines, they are more diffuse and irregular in shape than a contrail. Can an app be developed to help a ground observer determine the probability that an aircraft made the thin lines of white 'clouds' overhead?”

Contrail is one of the important resources of global warming. Contrails has been estimated to cause a tropospheric warming of 0.2 to 0.3 per decade by a general circulation model simulation of contrails (Minnis, Ayers, etc. 2004). Contrails reflect solar radiation and absorb and emit thermal infrared radiation, they will make a radiative forcing that depends on many factors, especially contrail optical depth and coverage (Sassen 1997). For scientists, knowing the contrails is an important way to know about climate changes.

Viewing the image to figure out the contrails are very inefficient, as on an image, there cloud be many other information to make detection of contrails difficult. This project is going to use MatLab to program a solution for this problem.

In this report, I will discuss the background of the projects, my solution and some results. Also the problems and future work we can do for this topic.



Background

Past Solutions

In the NASA space challenge website, many participants had offered their solution by machine learning or did their prediction by the flight timetable and route. However, this method usually only can be used to get the exist of contrails, or the possibility of contrails exist. Meanwhile, it really relies on the flight data and the photo location, but this cannot be convenient for scientists to connect the flight database.

Here are some examples:

- Contrailers-Exeter

Link: <https://2016.spaceappschallenge.org/challenges/aero/clouds-or-contrails/projects/contrailers-exeter>

The team mentioned in their explanation that they determine the probability by examining recent flights in the area and the air temperature.

- Hot on the Contrail

Link: <https://2016.spaceappschallenge.org/challenges/aero/clouds-or-contrails/projects/hot-on-the-contrail>

They do a machine learning solution, however, they can only do it for telling the possibility of contrails exist, but this solution doesn't know where contrails are.

This solution can will done the NASA space challenge, but it seems not useful for researches about contrails.

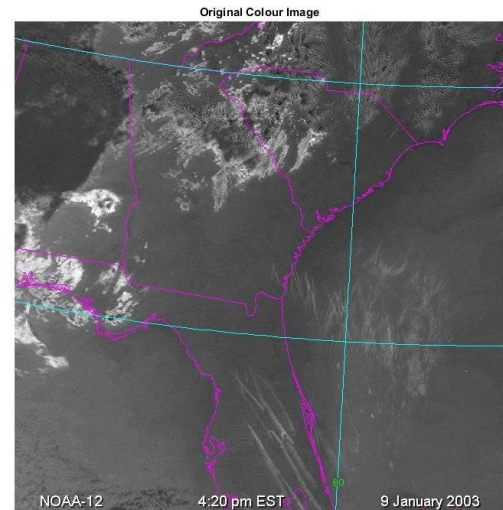
Aim and Target

For scientists to do a better research on contrails and clouds, the aim of this project is trying to make a program to automatically figure out the contrails from clouds on different kinds of images, such as satellites images, photos, and so on. After the contrails in the image figured out by the program, they will be highlighted and a comparison image will be output.

Scope and Constraints

1. This project will not be able to use for the bad resource with interference information, such as “figure1.jpg” (right figure) with latitude and longitude lines and map boards. Those lines are much clear than the clouds image, they will distribute the detection result.

2. This project may be inefficient to process large images. Our line bad pixel reduce algorithm is a ...



Approach

Solution Strategy


This project's target is distinguishing the contrails from the clouds, so we started by thinking the main difference between the graphs of contrails and clouds.

Contrails graph usually shows as a line on the image. Meanwhile, the clouds graph layouts in a random discrete distribution. In this way, we can make the program detect the straight lines in images to recognize the contrails.

However, it should be realized that even when the contrails look like pencil lines on image, there actually doesn't exist a really straight line when scanning through the image data. Because contrails also have some width on the image (in the real world, contrails could be several kilometers in width (CONTRAILS FACTS, page 3)). The real shape of contrails seems more like some long and thin rectangles with two fading sides. In this case, the problem became much harder.

Since this solution is hard to operate, instead of detecting the contrails' width, it is much easier to just detect the two clear sides of contrails. In this case, we can ignore the image inside the contrails, as well as other pixels inside the cloud by doing the edge detection.

After doing the edge detection, the results edge image can be detected lines by hough transform. Another problem was that there were too many edge



pixels in the image (see the result by ... below). Those bad edge pixels made the hough transform give bad lines which are not came from contrails.

To solve this problem, some processing on the edge image to get rid of bad edge pixels is necessary. In order to delete those bad edge pixels from clouds or other graphics, each pixel with a certain size of pixels around is checked as a small block. As lines can be divided into infinity small lines, if a line go through a small block, there must exist a small line inside the small block. We can use polynomial curve fitting to get the slope of small line, and delete the waste pixels. Then a much clean edge result will be got by putting all the processed small blocks into a new image.

After this processing, use the hough transform, a much better result will be got.

Here is the whole solution way:

1. Grayscale the original image
2. Use Canny edge detection to get the edge
3. Get the small blocks pixel by pixel, and do the polynomial curve fitting to get the small line information
4. Put all small line information into a new image
5. Do the hough transform on new image
6. Plot the results

MatLab Methods

■ Canny edge detection

```
Edge_image = edge (I, "canny");
```

Input: grayscale Image, I

Output: the black and white edge image

Canny edge detection is a technique to extract useful structural information from different vision objects and dramatically reduce the amount of data to be processed. There are following steps:

1. Apply the Gaussian filter to remove the noise;
2. Find the intensity gradient of the image;
3. Apply non-maximum suppression to get rid of spurious response to edge detection;

4. Apply double threshold to determine potential edges to filter out the edge pixel with the weak gradient value and preserve the edge with the high gradient value;
5. Track edge by hysteresis: Finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges.

■ Polynomial curve fitting

`p = polyfit(x,y,n)`

Input: x and y are vectors containing the x and y data to be fitted;

n is the degree of the polynomial to return;

Output: p is the third-degree polynomial that approximately fits the data.

This algorithm returns the coefficients for a polynomial $p(x)$ of degree n that is a best fit (in a least-squares sense) for the data in y . The coefficients in p are in descending powers, and the length of p is $n+1$.

■ Hough Transform and Hough Line Transform

`[H,theta,rho] = hough(BW);`

Input: the black and white image;

Output:

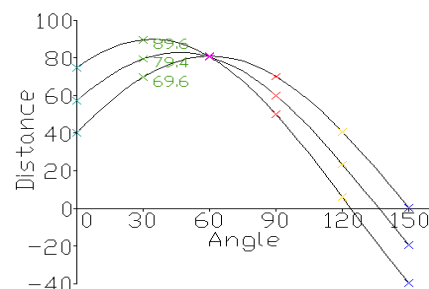
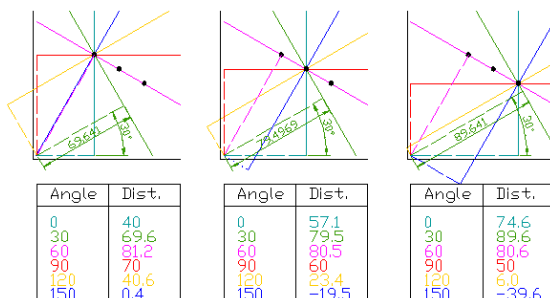
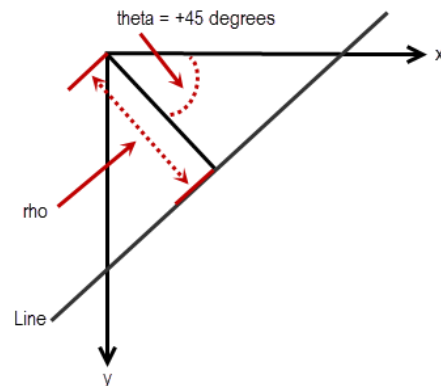
H is Hough transform matrix, returned as a numeric array;

Theta is the angle in degree between the x-axis and rho vector;

Rho is the distance from origin to the line along a vector perpendicular to the line;

The Standard Hough Transform (SHT) uses the parametric representation of a line: $\rho = x \cdot \cos(\theta) + y \cdot \sin(\theta)$.

For each pixel of the image, a numbers of lines going through it with all different angles. If they are in the same line, they will have the same rho and theta number. During the graph or different angle and distance, we will get a cross point, which means the straight line's rho and theta.



■ Median Filter

```
B = medfilt2(A, [m n]);
```

Input: A, the original image;

[m n], the isolate pixels' size to be deleted

Output: B, the image after deleted the isolate pixels.

Go through the whole image, if there is $m \times n$ pixels exist, then copy those information as `zeros[m x n]`.

Algorithm to reduce the bad edgels

This is a script, the input will be the edge image, and the output will delete the little block contains the small enough residual values.

Since the Canny edge detection gives many edgels we need to further process these edge maps to eliminate edgels (edge pixels) that are not parts of straight lines. For each $2*s+1$ by $2*s+1$ square neighborhood about a pixel, we fit all the neighborhood edgels to a straight line using `polyfit`. We fit either $y=mx+b$ for lines where $|m| \leq 45$ degrees and $(x=y-b/m)$ if $|m| > 45$ degrees. This takes care of horizontal and vertical lines. Now we have the equation of the best line fit for all the edgels in a neighborhood. But how good is this line fit? For each neighborhood edgel we compute the residual of that edgel's neighborhood fit to the line. We compute the overall residual as the square root of the sum of these squared residual values.

Neighborhood with poor line fits will have large overall residual values, we remove bad edgels using a threshold of 15 determined by trial and error.

Pseudo code for project

```
I = Read image;
grayscale (I);
image = Canny edge detect(I);
set the block size, height and width (s, height, width)

initial number no lines = 0;

for x = (1+s) to (height-s) {
  for y = (1+s) to (width-s) {
    initial points number = 0;
    if pixel is on an edge {
      for i = (x-s) to (x+s) {
        for j = (y-s) to (y+s) {
          if block pixel is edge {
            record them;
```

```

        points number ++;
    }
}
}
if at least one line in a block {
    m = polyfit (x and y data recoded);
    if (line's slope >1 or <-1) {
        compute the residual r ;
    }
    else {
        modify the m ;
        compute the residual r ;
    }
    save r values
}
else {
    number no lines ++;
}
}
}

get non_zero_r;
sort non_zero_r;
set the precentage p;
threshold = cast(p% * non_zero_r);

for x = (1+s) to (height-s) {
for y = (1+s) to (width-s) {
    if the r value is in the shreshold and non zero {
        write it on new_image;
    }
}
}

new_image = medfilter (new_image);
get hough transform matrix(H), theta(T) and rho(R) by
hough(new_image);
get peaks(P) = houghpeaks (new_image);
lines = houghlines(new_image, T, R, P);

for k = 1 to number of lines {
    plot lines on origin image(I);
}

```


Experimental Result

Figure1:

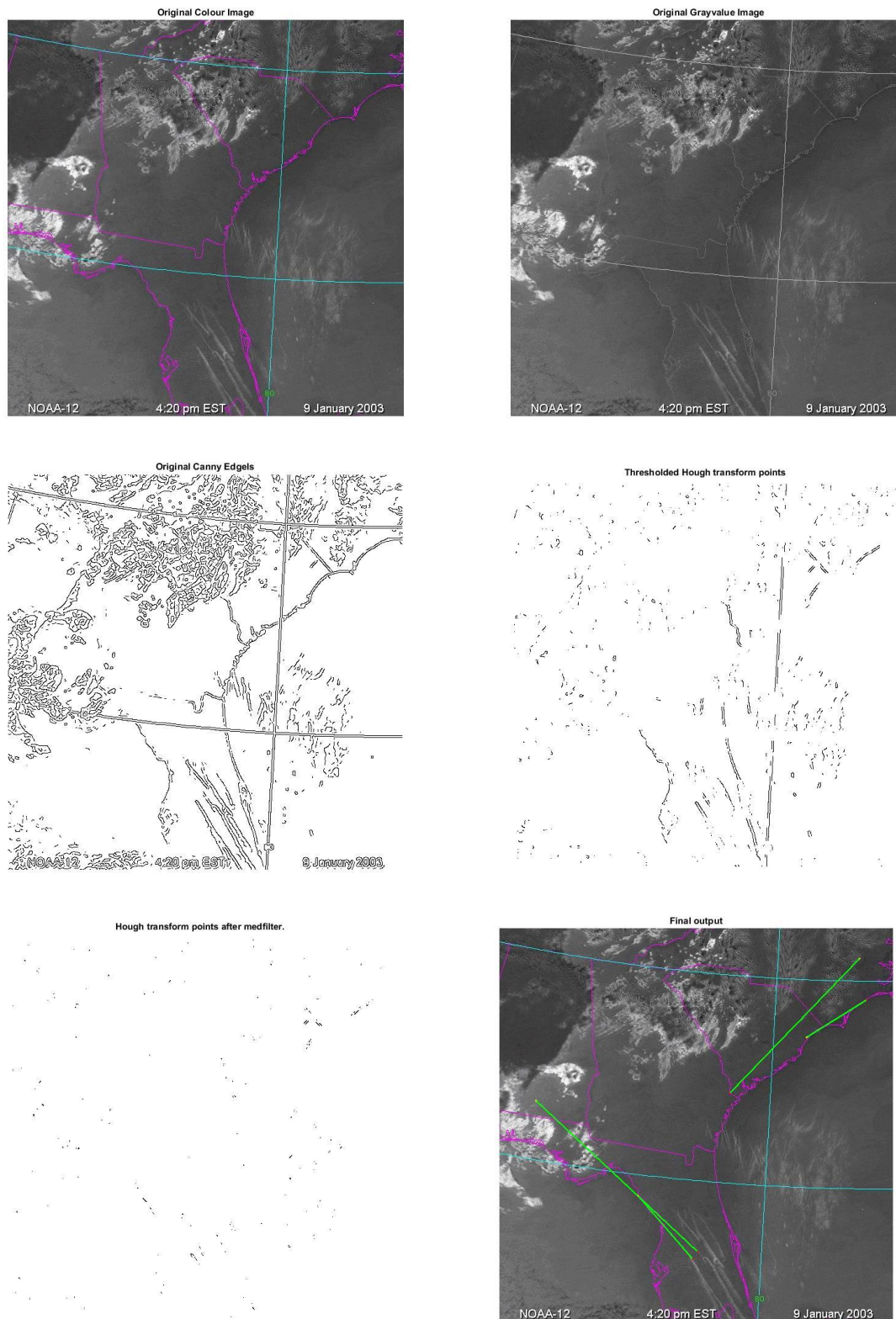


Figure2:

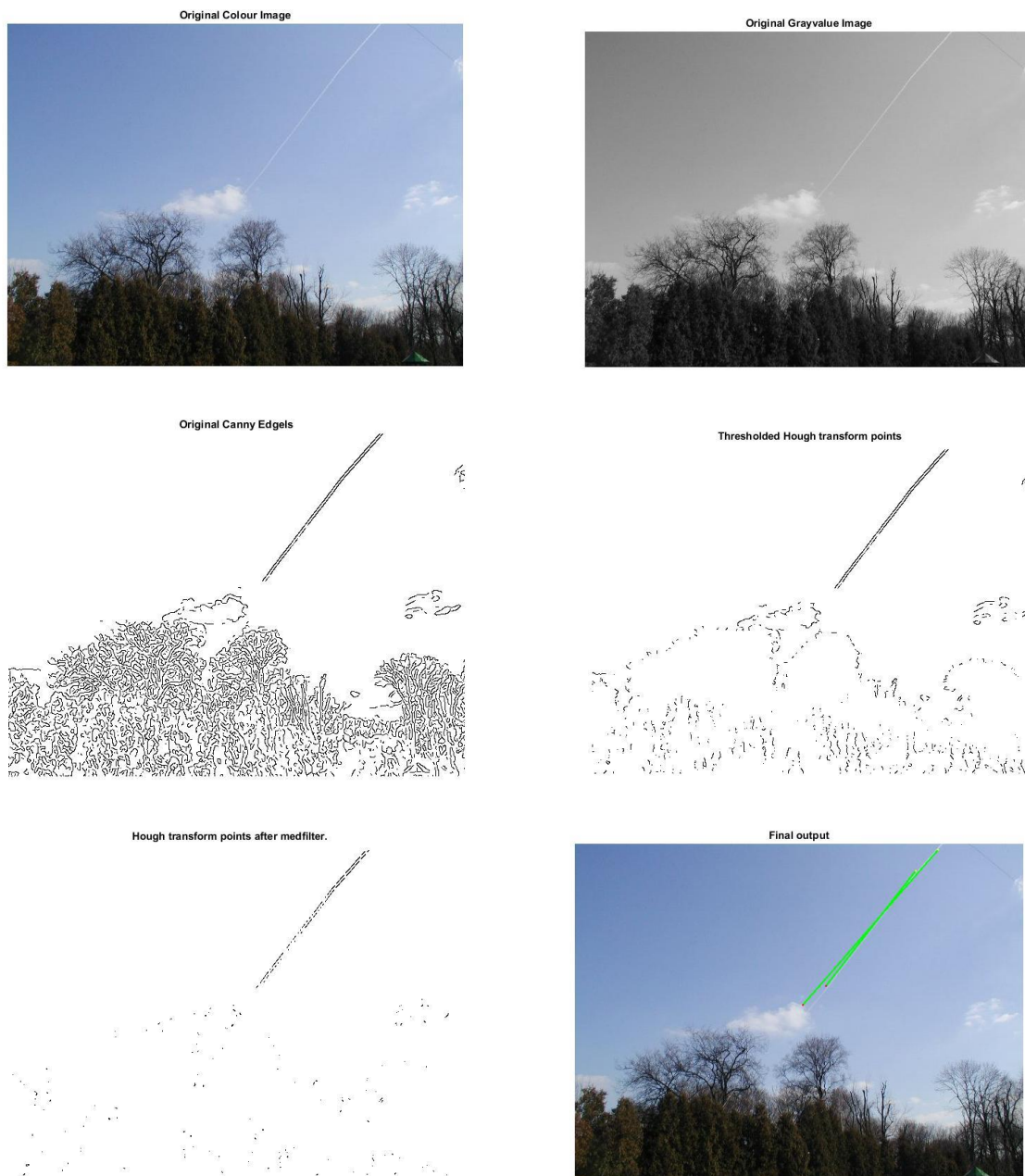
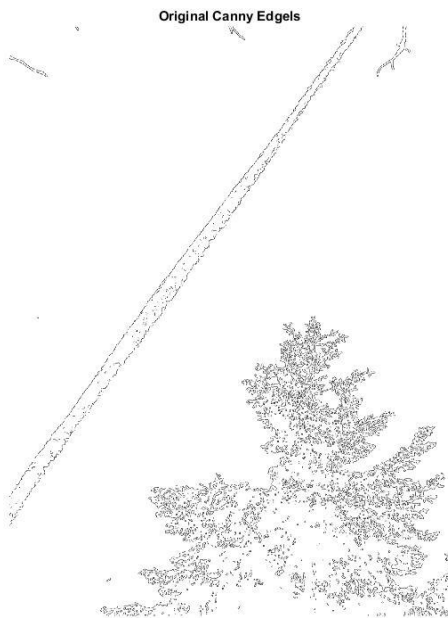
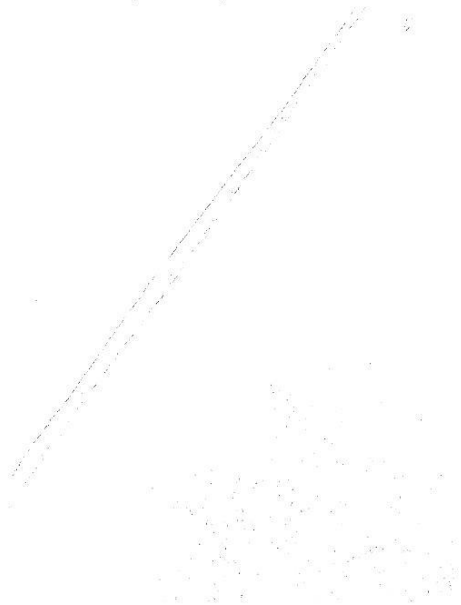


Figure3:





Hough transform points after medfilter.



Final output



Figure4:

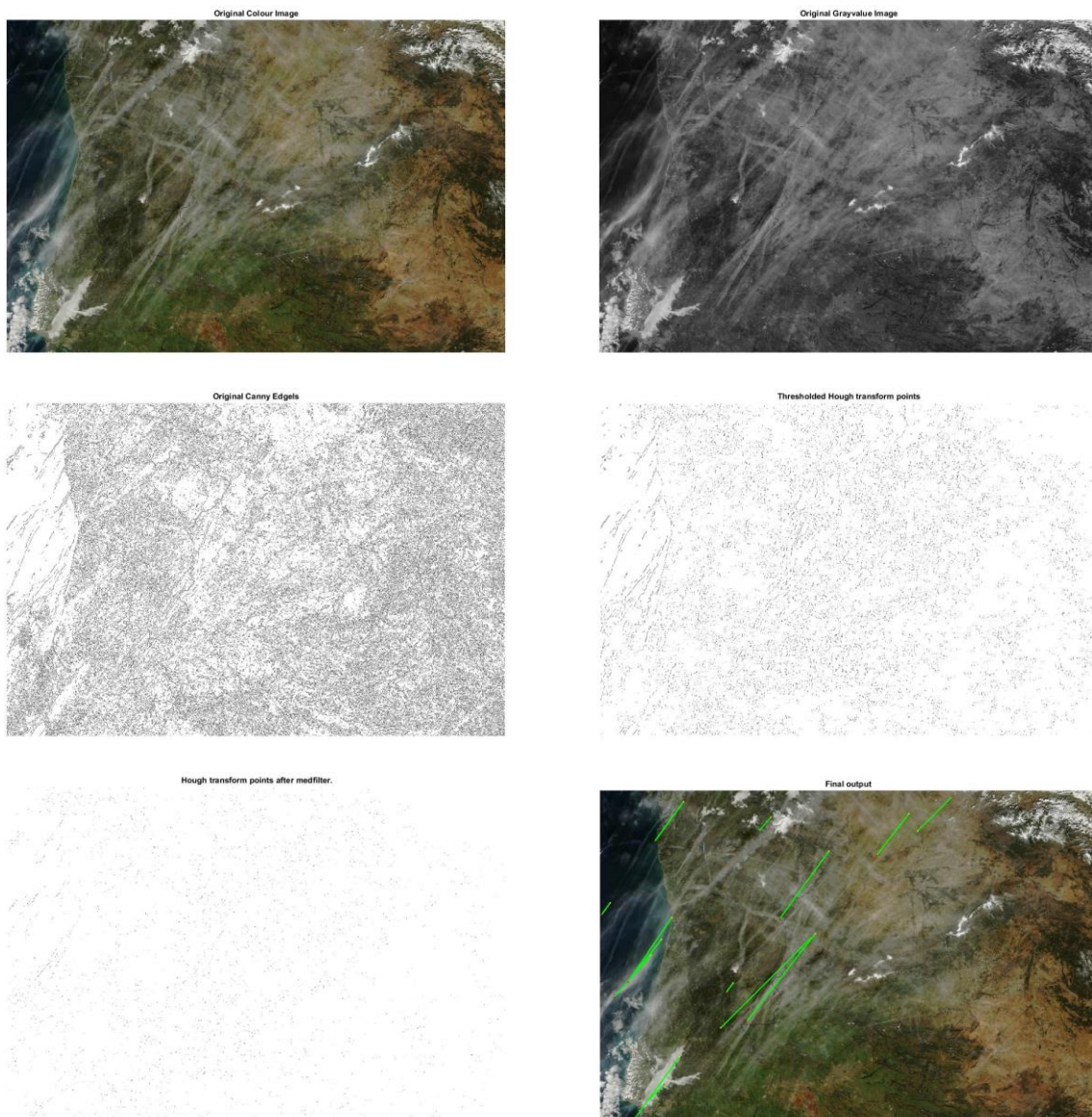


Figure5:





Problems

This project solution still has some problems.

6. Some parameters need to be manually set if the best result is needed. Such as the canny edge threshold; block size; the r value threshold; and hough transform minimum length, minimum gap, line numbers. Otherwise, the result would ignore some details when processing the multiple contrails image.

7. The image with multiple contrails cross each other will be not be processed well because of some contrails-cross block might be deleted because of too large r value.

8. Super large image will be processed very slow because of the pixel by pixel processing algorithm. For example, figure4's size is 2768×1845 , and it took 1902 seconds (31.7 minutes) to process it (./rst/figure4.txt).

Further Works

The further works is going to solve the problems exist in current solution. Here are two directions to improve the program to make it more efficient: easier to use and faster to process the image.

1. Automatically set some parameters, such as the parameters for canny edge detection. To solve this problem can make the program easier to use, also get more accurate result.

2. Change the next loop of reducing the bad pixels' algorithm. Because processing the whole image takes a $O(\text{size} * \text{block size})$ to finish the job, we may think a way to do it after we can narrow the image area we should process, then it can save a lot more time.



Conclusion

This project is designed to recognize the contrails from clouds image. Using canny edge detection, polynomial curve fitting, and hough transform to solve this problem. Following the algorithms below:

1. Grayscale the original image
2. Use canny edge detection to get the edge
3. By getting the small blocks pixel by pixel, and do the polynomial curve fitting to get the small line information
4. Put all small line information into a new image
5. Do the hough transform on new image
6. Plot the results

From the results, it can be seen that the problem has been fairly solved. However, still some problems exist, such as parameters setting, data miss deleted, and time issues.

We can still do some further research on solving this problem, for example, making the program more efficient or easier to use.



Resources

Reference

1. CONTRAILS FACTS." US Environmental Protection Agency, n.d. Web. <https://www3.epa.gov/otaq/regs/nonroad/aviation/afd-051013-001.pdf>
2. Minnis, P., Ayers, J. K., Palikonda, R., & Phan, D. (2004). Contrails, cirrus trends, and climate. *Journal of Climate*, 17(8), 1671-1685.
3. Sassen, Kenneth. "Contrail-cirrus and their potential for regional climate change." *Bulletin of the American Meteorological Society* 78.9 (1997): 1885-1903.
4. <https://2016.spaceappschallenge.org/challenges/aero/clouds-or-contrails>