Towards Finding Contrails in Cloud Images

Tuochaolong Zhang

Western University 2016

Towards Finding Contrails in Cloud Images

Tuochaolong Zhang

Thesis for
Computer Science 3380F - Project
Western University
Dr. John Barron

presented by
Tuochaolong Zhang

Octmber 15 2016

First Viewer: Dr. John Barron
Second Viewer: Dr. John Barron
Day of Oral test:

Contents

List of Figures

List of Tables

Summary

Here is a maximum one - sided summary of the Dissertation.

This is a new paragraph.

Chapter 1

Introduction

This project topic was originally posted by NASA Space Apps Challenges. The next paragraph is the introduction on the 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?
Contrails are potentially important sources of global warming. Contrails have been estimated to cause a tropospheric warming of 0.2 to 0.3 degree per decade by a general circulation model simulation of contrails (Minnis et al. 2004). Contrails reflect solar radiation and absorb and emit thermal infrared radiation. They 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 is very inefficient, as, in an image, there are other clouds data that make detection of contrails difficult. This project uses MatLab to program a solution for this problem. MatLab provides many built-in functions useful for image processing in general and this project in particular.
In this report, I will discuss the background of existing projects, my solution, and some results. Also, we give the conclusion and future work for this topic.

Chapter 2

Background

2.1 Past Solution

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, these methods usually only can be used to find the existence of contrails, or the possibility of those contrails exist. Meanwhile, it really relies on the flight data and the photo location, but it is not convenient for scientists to access all the flight database.

Here are some examples:

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

- Hot on the Contrail
  They do a machine learning solution, however, they only determine the possibility of contrails existing, but not where contrails are. This solution has been done the NASA space challenge, but it seems not useful for detecting the actual contrails locations.

2.2 Aim and Target

In order for scientists to do better researchers on contrails and clouds, the aim of this project proposes a program to automatically segment the contrails from clouds on different kinds of images, such as satellites images, photos, and so on. After the contrails in the image have been determined, they are highlighted and a comparison image is outputted.

2.3 Scope and Constraints

- This project will not be able to use the images with interference information, such as figure1.jpg (figure below) which has latitude and longitude lines and map boards. Those lines have much clearer border than the clouds image, and they will disrupt the detection result.

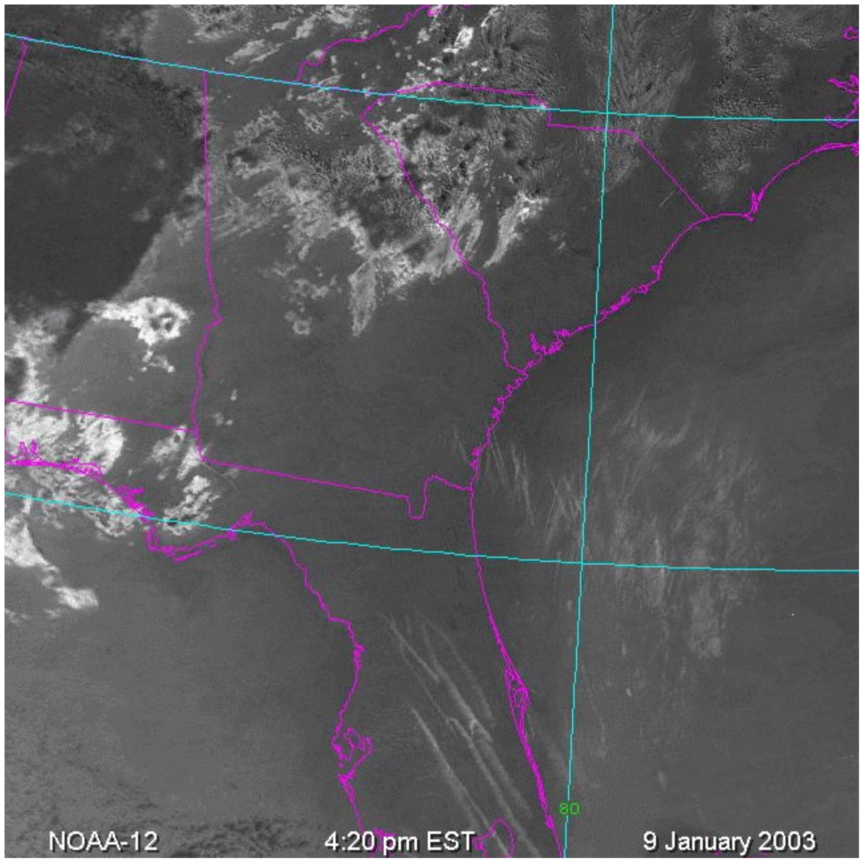


Figure 2.1: "figure1.jpg"

- Our program may be inefficient to process large images. The strategical removal of non-line edgels is complicatedly expensive.

Chapter 3

Approach

3.1 Solution Strategy

This projects goal is to distinguish contrails from the clouds, so we started by thinking about the main difference between the graphs of contrails and clouds.
Contrails usually show as a line on the image. Meanwhile, the clouds layout in a random discrete distribution. In this way, we can write a program to 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 the image, there doesnt exist an actual straight line when scanning through the image data. Contrails also have some width in 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 detection problem became much harder.
Since this solution is hard to compute, 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 performing the edge detection, we can detect lines by Hough Transform on the edgel map. Another problem was that there were too many edge pixels in the image (see the result by below). Those bad edgels came to the Hough Transform to give incorrect lines, which do not come 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 goes 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 the best small line in each block. Then we check how well the block edgels fit the computed line. Poor fits not as determined by a line residue causes the create edgel on the block to be rejected from the further Hough Transform.
After this processing, use the Hough Transform, a much better result will be got.

Here is the whole solution:

1. Convert the color image to gray-scale;

2. Use the 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 the edgels surviving the polynomial curve fitting procedure into a new image;

5. Do the Hough Transform on new image;

6. Median filter the new image;

7. Plot the results.

3.2 MatLab Methods

3.2.1 Canny Edge Detection

Edge_image = edge (I, 'canny');

Input: I, grayscale Image
Output: Edge_image, 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.

3.2.2 Polynomial curve fitting

p = polyfit(x,y,n);

Input:
x and y, vectors containing the x and y data to be fitted;
n, the degree of the polynomial to return;
Output: p, the third-degree polynomial that approximately fits the data.

This algorithm returns the coefficients for a polynomial p(x) of degree n that is the 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.

3.2.3 Hough Transform and Hough Line Transform

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

Input: BW, the black and white image;
Output:
H, the Hough Transform matrix be returned as a numeric array;
theta(θ), the angle in degree between the x-axis and rho vector;
rho(ρ), the distance from origin to the line along a vector perpendicular to the line;

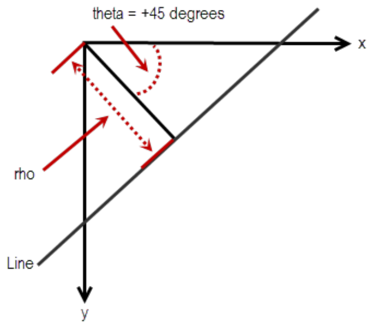
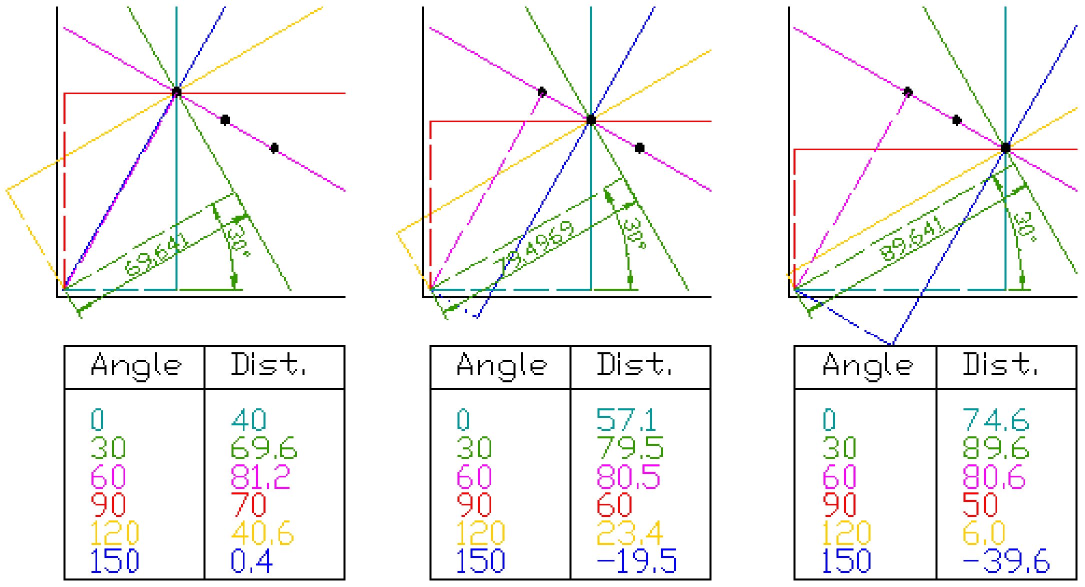


Figure 3.1: theta(θ) and rho(ρ) of Hough Transform

The Standard Hough Transform (SHT) uses the parametric representation of a line:

$$\rho = \mathrm{x}\cos\theta + \mathrm{y}\sin\theta.$$

For each pixel of the image, 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 lines rho and theta.
As we can see from above figure (Wikipedia.com), when the Hough Transform algorithm goes through all different angles (θ) in every pixels, there is a certain angle ($\theta = 60$ in above figure), it makes the perpendicular distance from the line to origin keeps the same ($\rho = 81.2 \ = 80.5 \ = 80.6$). When we draw all the information below, we can have a much clear view for how the Hough Transform works.



Angle	Dist.
0	40
30	69.6
60	81.2
90	70
120	40.6
150	0.4

Angle	Dist.
0	57.1
30	79.5
60	80.5
90	60
120	23.4
150	−19.5

Angle	Dist.
0	74.6
30	89.6
60	80.6
90	50
120	6.0
150	−39.6

Figure 3.2: θ=60, ρ are same for the 3 points

3.2.4 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 isolated pixels.

Put m x n pixel values around each target pixel into an array, then order all the values in the array, after this, we find the median value of the array and put it into the target pixels.

3.3 Algorithm to reduce the bad edgels

3.3.1 Describe

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 $2 \times s$+1 by $2 \times s$+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=$\frac{y-b}{m}$ if $|m| \geq 45$ 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 edgels neighborhood fit to the line. We compute the overall residual as the square root of the sum of these squared residual values.
Since neighborhoods with poor line fits will have large overall residual values, we remove bad edgels using a threshold of 15 determined by trial and error.

3.3.2 Pseudo code

```
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:
                        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);
```

Chapter 4

Result

4.1 Best Result and Parameters

4.1.1 figure1

4.2 Discussion

4.3 Effect of Parameters

4.4 Problems

This project solution still has some problems:

1. Some parameters need to be manually set if the best result is to be obtained, such as the canny edge threshold; the block size; the r value threshold; and Hough Transform minimum length, minimum gap, line numbers. Otherwise, the results would surprise some details when processing the multiple contrails images.

2. The images with multiple contrails that cross each other will be not be processed well because some blocks which have multiple contrails image cross each other might be deleted because of too large r value.

3. Super large image will be processed very slow because of the pixel by pixel processing algorithm. For example, figure4s size is 2768 x 1845, and it took 1902 seconds (31.7 minutes) to process it (details see this project package /rst/figure4.txt).

Chapter 5

Future Work

Further work is needed to solve the problems that 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, block size, or r value threshold, etc.

2. Use the MatLabs parallel processes computing to reduce the loop processing time.

Chapter 6

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 pixels survive the block processes 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.

Appendix A

The first appendix

Hier steht der erste Anhang.

Appendix B

The Second appendix

Hier kommt der zweite Anhang.

thanks

Danke.

vitae