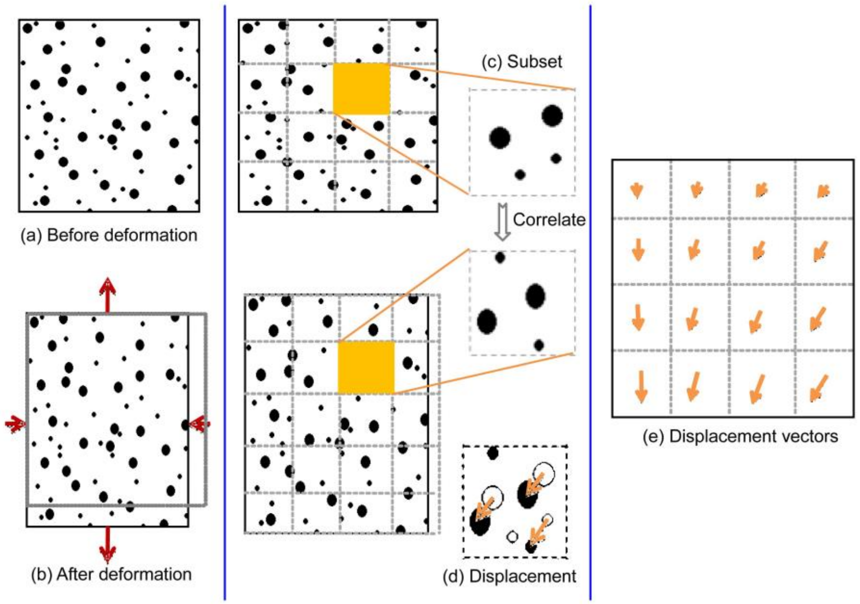
Digital Image Correlation With Matlab

Sanders Li

Digital Image Correlation and Its Applications

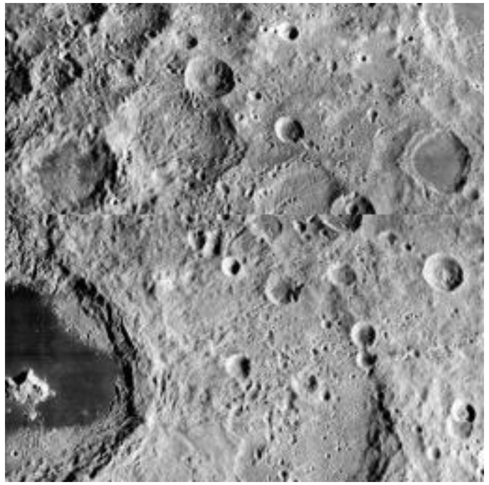
- Digital images are made of a matrix of numbers, correlating to the brightness of the pixel
- Accessibility
- Physical objects can be compared in a non-invasive way
- Can be used in fields such as material science to calculate minute deformations

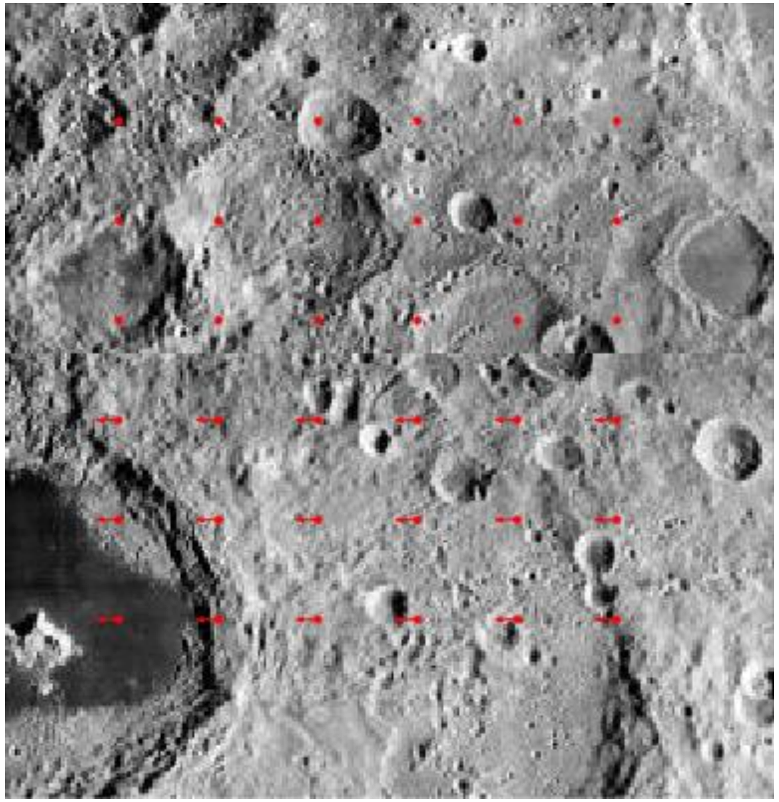


http://www.jzus.zju.edu.cn/iparticle.php?doi=10.1631/jzus.A1200274

Objective

Given two images, img_a and img_b, quantify and visualize the displacement field using digital image correlation





Methodology

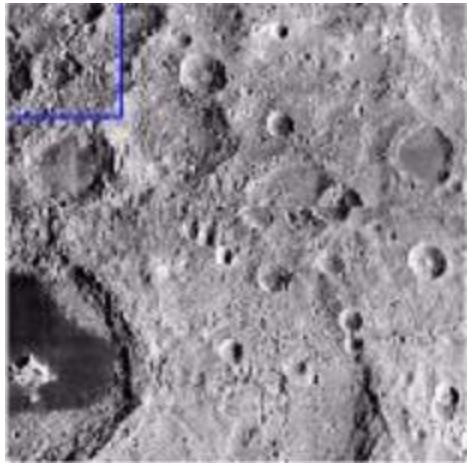
- Choose a pattern box A in img_a
- Choose a search box S in img_b sufficiently large to find A within S

  This is done to improve the speed of the program, but relies on the assumption that A is within S (small displacement)

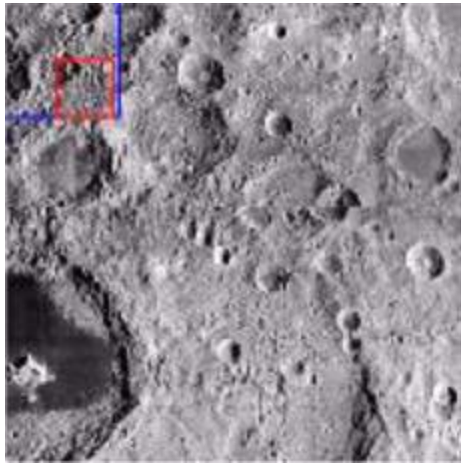
- Within S, choose some pattern box B the same size as A

A in img_a

S in img_b



B in S in img_b

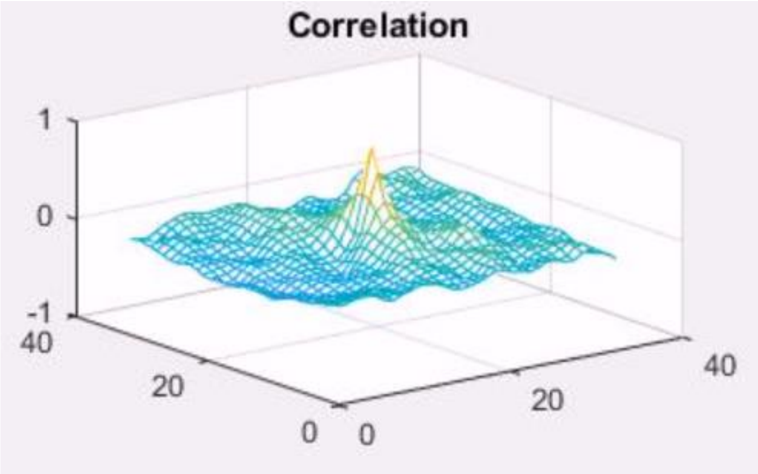


Methodology

- Using the following algorithm, create a correlation chart and find where B best matches A

$$C_{i,j} = \frac{\sum\sum(B_{i,j} - B_{avg})(A_{i,j} - A_{avg})}{\sqrt{\sum\sum\left(B_{i,j} - B_{avg}\right)^2 \sum\sum\left(A_{i,j} - A_{avg}\right)^2}}$$

- Shift S and A and repeat as much as possible



Initialization

- Retrieve the image, throw it in working directory

  For Visualization: Initiate creation of a movie object. Display the image on a figure.

- “Flatten” the image for easier processing

```
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  specify the working directory where the a/b image pair is stored
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
working_dir = 'F:\images\Control\';  % keep the "\" at the end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Get the filenames of the a/b image pair in working_dir.  This will create
% a structure where file names are images(1).name for img_a and,
% images(2).name for img_b.
images = dir([working_dir,'img*.jpg']);
```

```
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Load img_a and img_b and Pre-Process image pair:
%  flatten to 1D (grayscale) if the images are multidimensional (e.g., RGB)
%  This block uses the function 'flatten_rgb_image' which we provided; make
%  sure this function is in your Matlab path.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if createMovie == true
    movie = VideoWriter('visualizer.avi');
    open(movie);
end;

for i = 1:length(images)
    % load image data into a temporary variable
    temp = imread([working_dir,images(i).name]);
    % show original image
    if setup_mode == 1
        figure(1);  subplot(1,2,i);
        imshow(temp);  title(images(i).name,'Interpreter','none'); hold on;
    end

    % Assign flattened (grayscale) image data to the structure "IMAGES";
    % this data will be used for interrogation.
    images(i).data = flatten_rgb_image(temp);
end
% End of image Pre-Processing setup
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Flatten

- Our goal here is to reduce a RGB image to greyscale
- Average each R, G, B intensity value at each pixel
- Convert to double for some arithmetic
- Convert back to uint8 format, insert in matrix to produce greyscale image

```
% if image data is RGB (MxNx3), "flatten" to grayscale (MxNx1)
[M, N, D] = size(img);   % rows , columns , image_depth

if D > 1  % flatten image
    % initialize MxNx1 matrix
    K = zeros(M,N,1);
    % sum all layers (RGB colors) together, then calc. the mean intensity
    for j = 1:D
        % Convert Integer image data to a Double for arithmetic
        K = K + ( double(img(:,:,j)) + 1 );
    end

    % calculate the mean intensity at each pixel (divide by D pixels)
    K = K./D;

    % convert back into unsigned 8-bit integer format (uint8)
    imgGS = uint8( round(K - 1)); % added ROUND per Matlab suggestion, but

else
    % image data was already 1D, just assig to output variable 'imgGS'
    imgGS = img;
end
```

Initialize subimage and search box sizes

- Calculate size of sample subimages **A** and **B**
- Calculate search box size **S**
- Calculate where to start **A** and **B**
- Calculate how much to shift **A**, **B**, and search box **S**
- Calculate number of times to shift in the x and y directions

```
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Correlation Parameters for the image pair:
%  image box and search box dimensions (in pixels);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[img_m,img_n] = size(images(2).data);

Bx = floor(img_m/8);
By = floor(img_n/8); %size of A and B

Sx = floor(Bx*1.75);
Sy = floor(By*1.75); %search box size

startx = floor(Sx/2);
starty = floor(Sy/2); %where to start A, min 1 (start at index 1)

shiftx = Sx;
shifty = Sy;

nx = floor((img_m-Sx-startx)/shiftx)+1; %number of times to shift x
ny = floor((img_n-Sy-starty)/shifty)+1; %number of times to shift y
```

Visualization of image boxes

- Set up a figure, titles, and images in respective places for visualization
- Matrices are created to hold the data points used to draw boxes on the images in the figure

```
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  For each image box in img_a, calculate the displacement.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% nx*ny = total number of displacement calculations (grid points).  This is
% a function of the image size and the Correlation Parameters from above.

hFig = figure(2);
set(hFig, 'Position', [500 200 1280 720]);
hold on;
subplot(3,3,1); imshow(uint8(images(1).data));
title(['All A in ' images(1).name],'Interpreter','none'); hold on;
subplot(3,3,2); imshow(uint8(images(1).data));
title(['Current A in ' images(1).name],'Interpreter','none'); hold on;
subplot(3,3,4); imshow(uint8(images(2).data));
title(['All S in ' images(2).name],'Interpreter','none'); hold on;
subplot(3,3,5); imshow(uint8(images(2).data));
title(['Current S and B in ' images(2).name],'Interpreter','none'); hold on
subplot(3,3,7); imshow(uint8(images(2).data));
title('Vectors','Interpreter','none'); hold on;

bdraw = [];
Abox = [];
Sbox = [];

x = zeros(nx,ny);
y = zeros(nx,ny);
u = zeros(nx,ny);
v = zeros(nx,ny);
```

Code - Main

- Loop through number of horizontal and vertical shifts

  Create a progress indicator

- Preallocate space for a correlation matrix **C** for our algorithm later in the code

- Define pixel array **A**

  *For Visualization, draw **A** on figure (for visualization)*

```
for p = 1:nx %y-dir for S
    % progress indicator...
    clc;
    fprintf('Progress: %.0f%%... \n',100*(p-1)/nx) %change from nx to ny
    for q = 1:ny %x-dir for S
        fprintf('\t%.0f%%... for row\n',100*(q-1)/ny)

        C = zeros(Sx-Bx+1,Sy-By+1);
        C = C-1; %create array of -1s

        % pixel array A
        Axpos = startx+Bx/4+(p-1)*shiftx;
        Aypos = starty/2+By/4+(q-1)*shifty;
        A = double(images(1).data(Axpos:Axpos+Bx,Aypos:Aypos+By));
        % specify array indices and convert to a double
        % NOTE: imshow does not like doubles,
        % so imshow(uint8(A)) will display A nicely
        A_avg = 1/(Bx*By)*sum(sum(A));

        figure(2); subplot(3,3,3);  imshow(uint8(A)); title('A');
        figure(2); subplot(3,3,1);
        rectangle('Position',[Aypos-1,Axpos-1,By,Bx],...
            'EdgeColor','r');
```

Code - Main

- Define pixel array **S**
- Draw **B** within **S**

  For visualization, write the current state of the figure to the movie as a single frame

```
% Find the displacement of A by correlating this pixel array with all
% possible destinations B(K,L) in search box S of img_b.
Sxpos = startx/2+(p-1)*shiftx;
Sypos = starty/2+(q-1)*shifty;
S = double(images(2).data(Sxpos:Sxpos+Sx,Sypos:Sypos+Sy));

figure(2); subplot(3,3,4);
rectangle('Position',[Sypos-1,Sxpos-1,Sy,Sx],...
    'EdgeColor','b');
figure(2); subplot(3,3,5); delete(Sbox);
Sbox = rectangle('Position',[Sypos-1,Sxpos-1,Sy,Sx],...
    'EdgeColor','b');
if animate == true
    drawnow;
end;
```

```
for i = 1:(Sx-Bx+1) % x pixel shift within S
    for j = 1:(Sy-By+1) % y pixel shift within S
        %tic %timer function, used to estimate time
        % pixel array B
        % specify array indices within S and convert to a double
        B = double(S(i:Bx+i,j:By+j));
        Bxpos = Sxpos+i-1;
        Bypos = Sypos+j-1;
        delete(bdraw);
        figure(2); subplot(3,3,5);
        bdraw = rectangle('Position',[Bypos-1,Bxpos-1,By,Bx],...
            'EdgeColor','r');
        if animate == true
            drawnow;
        end;
        figure(2); subplot(3,3,6); imshow(uint8(B)); title('B');
        if animate == true
            drawnow;
        end;
        if createMovie == true
            frame = getframe(gcf);
            writeVideo(movie,frame);
        end;
```

Correlation Algorithm

Core of the code

$$C_{i,j} = \frac{\sum\sum(B_{i,j} - B_{avg})(A_{i,j} - A_{avg})}{\sqrt{\sum\sum(B_{i,j} - B_{avg})^2 \sum\sum(A_{i,j} - A_{avg})^2}}$$

For each pixel array **B** within space **S** and find the correlation value (between 0-1) between pixel array **A** and **B.** Store this in matrix **C**

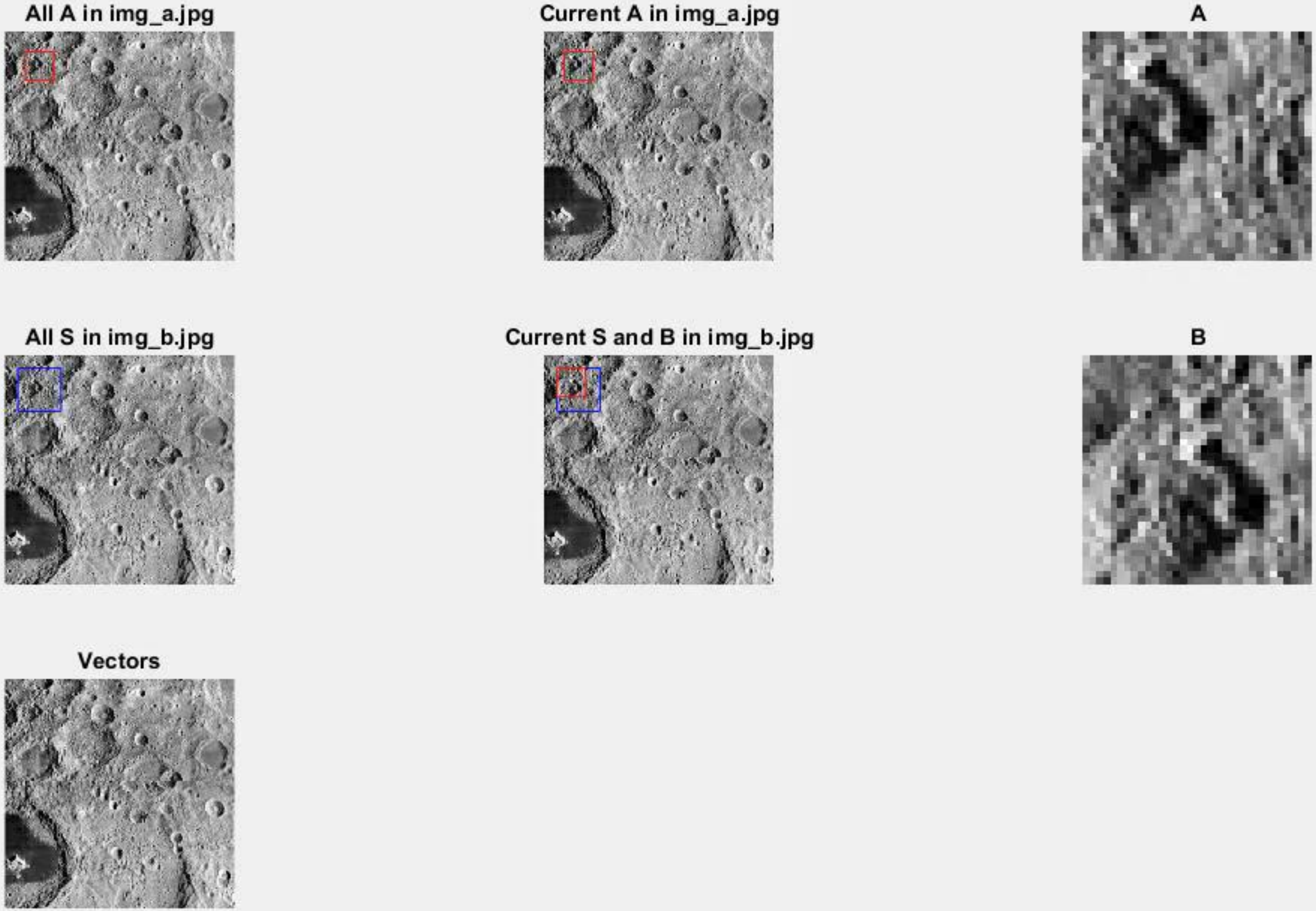
```
% Calculate the correlation coefficient, C, for this pixel array.
% Evaluate C at all possible locations (index shifts I,J).
% The best correlation determines the displacement of A into img_b.
%  Note: Double sum below effectively implements Double Riemann sum across k and l in lecture

B_avg = 1/(Bx*By)*sum(sum(B));
C(i, j) = sum(sum( (A - A_avg).*(B - B_avg) ))/...
          sqrt(sum(sum( (A - A_avg).^2 ))*sum(sum( (B - B_avg).^2 )));
figure(2); subplot(3,3,8); mesh(C); axis([0 Sy-By+1 0 Sx-Bx+1 -1 1]); title('Correlation','Interpreter','none');
```

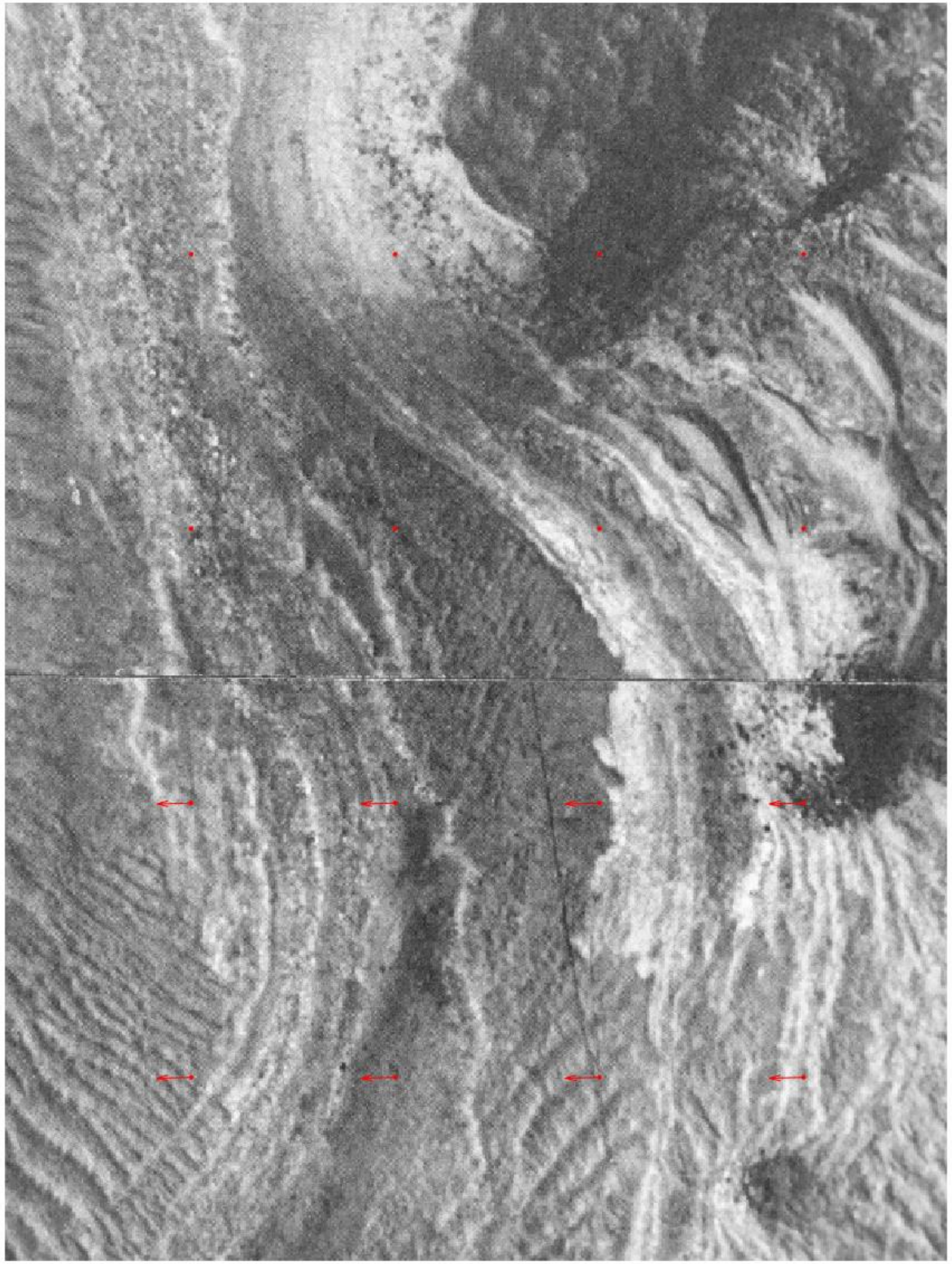
Code - Main

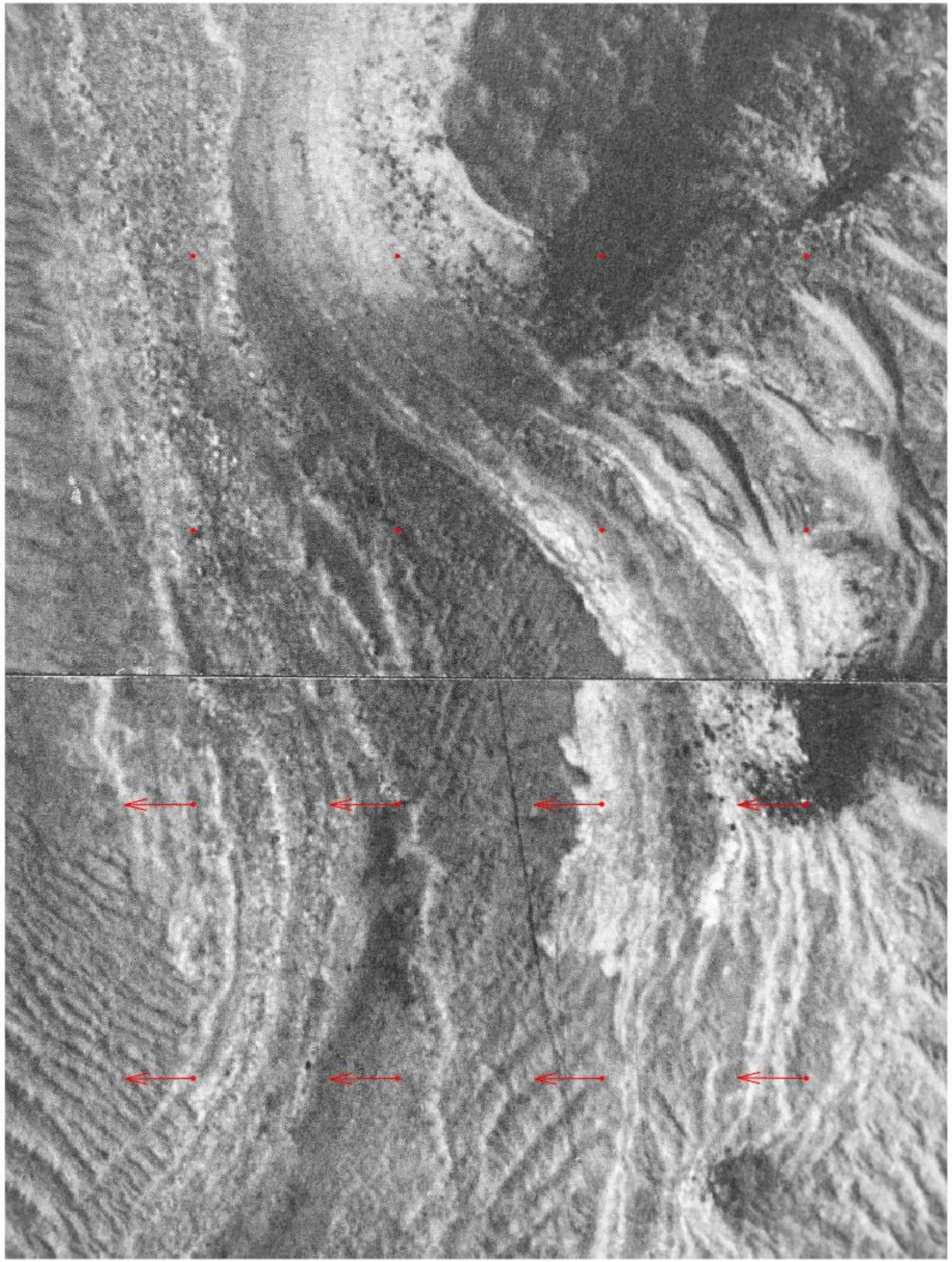
- Find the max value of **C**
- Translate that index position to variables **u** and **v** used to draw the vector

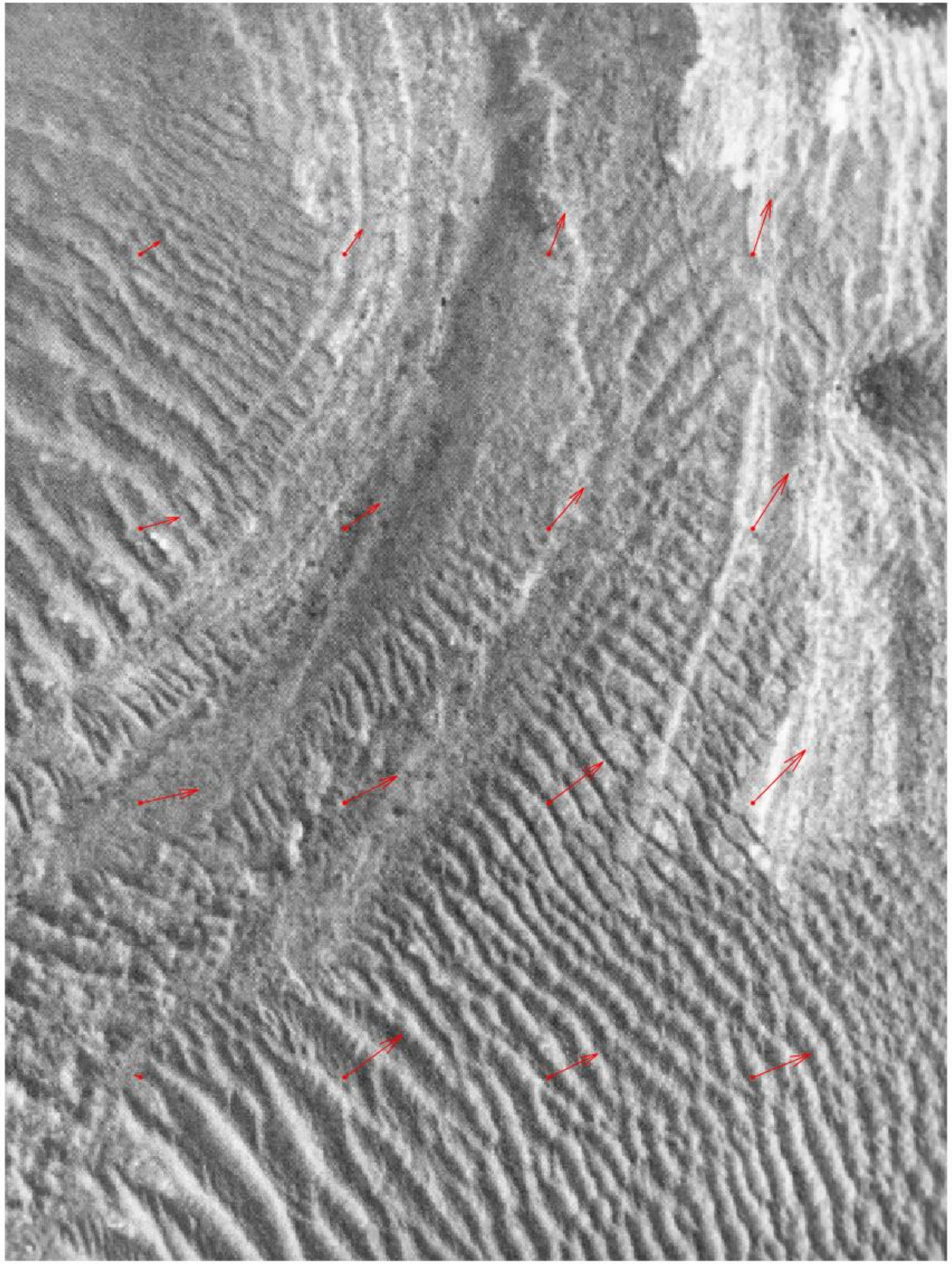
```
[maxCval1 maxCrow] = max(C);
[maxCval2 maxCcol] = max(maxCval1);
maxCx = maxCrow(maxCcol);
maxCy = maxCcol;
%maxCval2
x(p,q) = (Axpos-1 + (Axpos+Bx-1))/2;
y(p,q) = (Aypos-1 + (Aypos+By-1))/2;
u(p,q) = (Sxpos+maxCx-1)-Axpos; %x-displacement
v(p,q) = (Sypos+maxCy-1)-Aypos; %y-displacement

figure(2); subplot(3,3,7); quiver(y,x,v,u,'Marker','.','Color','r'); drawnow;
```

Process Visualization:









Future Improvements

FFT greatly reduces time complexity of algorithm

Object oriented

Multithreading

Support for RGB correlation