Introduction: Project 1

Author: Jordan Yono

Date: 03/01/2021

Course: ELEE-5920 Image Processing and Computer Vision

Github: https://github.com/jyono/elee-5920-image-processing

This project was quite doozy, but I learned a lot. I will leave this short and sweet because the rest of the project is quite lengthy :).

Part 1: Handling Image Arrays

a: Load images and get size

Load the images red_cow.jpg and snapper.jpg into Matlab. What are the sizes of the two images?. What method did you use to determine the size?

```
disp("<strong> Part 1, Section a");
```

```
Part 1, Section a
```

```
% Load images
redCowFile = imread("red_cow.jpg");
snapperFile = imread("snapper.jpg");

% Get image sizes
redCowFileSize = size(redCowFile); % 511x342x3 uint8
snapperFileSize = size(snapperFile); % 342x511x3 uint8

% Display whos breakdown in workspace
disp("<strong> Red Cow File Size");
```

```
Red Cow File Size
```

```
whos redCowFile;
```

```
  Name             Size                Bytes  Class    Attributes

  redCowFile     511x342x3            524286  uint8
```

```
disp("<strong> Snapper File Size");
```

Snapper File Size

```
whos snapperFile;
```

```
  Name              Size                Bytes  Class    Attributes

  snapperFile     342x511x3            524286  uint8
```

Method

This was pretty straightford. I utilized the matlab built in `imread` and `size` functions to load and size the images.

Results

The results were what I expected them to be. In this case, I think they are completely correct.

Conclusion

There is not much of a conclusion to draw here given the simplicity of the problem.

b: Color values at (r29, c86) for red_cow and (r198, c201) for snapper

What are the color values of pixels at row 29, column 86 for red_cow and row 198, column 201 for snapper? Please pay attention to image and matrix indexing in matlab.

```
disp("<strong> Part 1, Section b");
```

Part 1, Section b

```

% Get Value of specific pixels
redCowR29C86Pixel = impixel(redCowFile, 29, 86);
snapperR198C201Pixel = impixel(snapperFile, 198, 201);

%% Optionally used to manually verify results
%% of the above functions.
% imtool(redCowFile);
% imtool(snapperFile);

% Display rbg value
disp("<strong> Red Cow (R29,C86)")
```

Red Cow (R29,C86)

```
disp(redCowR29C86Pixel); % 180, 174, 150
```

```
   180   174   150
```

```

disp("<strong> Snapper (R198,C201)")
```

Snapper (R198,C201)

```
disp(snapperR198C201Pixel); % 43, 21, 7
```

```
    43    21     7
```

Method

For this problem, I used the `impixel` function. This function allows you to give the row and column values and returns the value of the selected pixel.

Results

The results were what I expected them to be. In this case, I think they are completely correct. I was able to verify them additionally by loading the image and looking at the specific pixel via `imtool` and viewing the value.

Conclusion

There is not much of a conclusion to draw here given the simplicity of the question.

c: Plot image showing profile and 3d plot rotated

Plot the 103rd row of the green plane for red_cow {Note: red = A(:,:,1), green = A(:,:,2), blue = A(:,:,3)}.Plot the 69th column of the blue plane of snapper. [Use the plot() command].

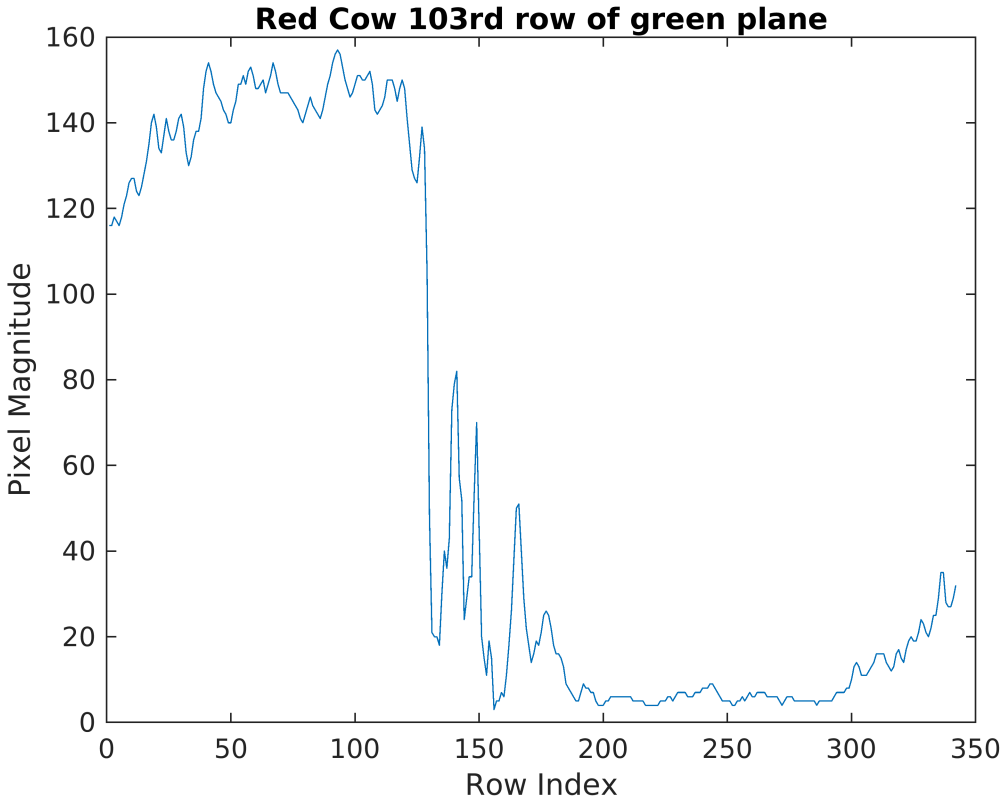
Each plot should place the pixel magnitude along the y-axis and the position along the row or column along the x-axis. Be sure to properly label the plots (using label, title etc).

```
disp("<strong> Part 1, Section c");
```

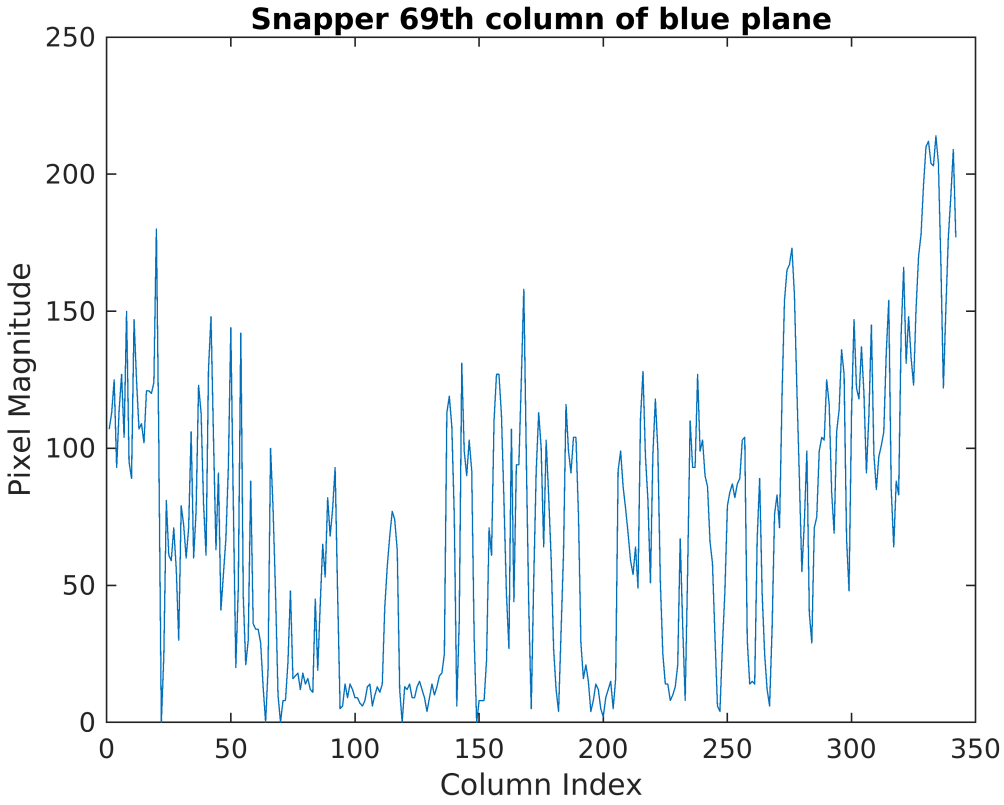
Part 1, Section c

```
% Plot the images
%
% Note:
% since Red Cow is 511x342 and Snapper is 342x511
% and we use Red Cow rows and Snapper columns,
% the x-axis for both will be 0-341

% Plot 103rd row, all columns, and z = 2(green)
plot(redCowFile(103, :, 2));
title('Red Cow 103rd row of green plane');
xlabel('Row Index');
ylabel('Pixel Magnitude');
```




```
% Plot all rows, 69th column, and z = 3(blue)
plot(snapperFile(:, 69, 3));
title("Snapper 69th column of blue plane");
xlabel('Column Index');
ylabel('Pixel Magnitude');
```

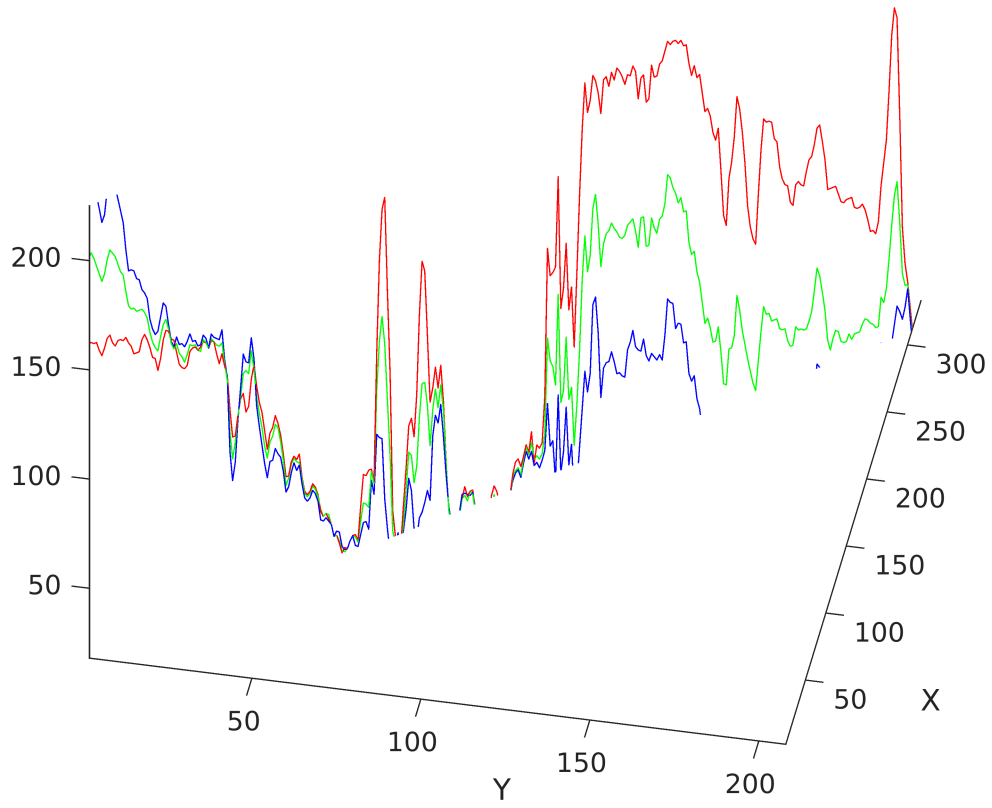


Next -- use the matlab intensity profile commands to select a 2-segment line to profile [Using matlab help -- search for information on creating an intensity profile of an image]. Plot the image showing the profile and the 3D plot rotated to make the results easy to interpret.

```
% Use image profile to draw line segment and rotate to view nicely.
imshow red_cow.jpg;
```


```
improfile(redCowFile, [1 511 1 511], [1 342 1 342]);

xlim([2 333])
ylim([2 208])
zlim([18 225])
view([-79 45])
```



Method

For this problem, I used the `plot` command to explicitly plot the given 103rd row of the cow's green plane and 69th column of the snapper's blue plane. Additionally, I used the `improfile` command to create a 2-segment line profile of the red cow.

Results

The results were what I expected them to be. In this case, I think they are completely correct.

Conclusion

Its very interesting to look at the cow's green plane. You can see that the grass in the background leads to a significant rise in intensity for those rows.

d: Create image in figure 1

Create and draw the image shown below in Figure 1. The image subblock (of size 128x128) on the top-left corner comes from snapper.jpg beginning at coordinates (128,192). The rest of the image is from red_cow.jpg. Use imshow() to draw the image. Hint (remember this is a color image).

```
disp("<strong> Part 1, Section d");
```

Part 1, Section d

```
redCowFileCopy = redCowFile;
%% Simple way to draw without imshow().
% left side says where on the cow to replace with snapper
redCowFileCopy(1:129,1:129,:) = snapperFile(128:256,192:320, :);
disp("<strong> Cow + Snapper 1");
```

Cow + Snapper 1

```
imshow(redCowFileCopy);
```


```
%% draw with imshow directly over existing image.
redCowFileCopy2 = redCowFile;

disp("<strong> Cow + Snapper 2");
```

Cow + Snapper 2

```
imshow(redCowFileCopy2);
% Retain existing plots
hold on
% Put cropped turtle in top left corner.
imshow(snapperFile(128:256,192:320,:));
hold off
```

Method

For this problem, I took the values of the snapperfile that we are interested in, and replaced the top left of the rec cow with the snapper values. Additionally, I did this an alternative way by putting the snapper image directly over the cow image.

Results

The results were what I expected them to be. In this case, I think they are completely correct.

Conclusion

It was interesting looking at this problem and thinking about two possible solutions. This could be achieved by either replacing the existing pixels, or simply placing the turtle on top of them.

e: Min/Max of grayscale versions of red_cow and snapper

What are the maximum and minimum values of the pixels in grayscale versions of images red_cow and snapper?

```
disp("<strong> Part 1, Section e");
```

```
Part 1, Section e
```

```
% Load the cow images and convert to grayscale
redCowFileGrey = rgb2gray(redCowFile);
snapperFileGrey = rgb2gray(snapperFile);

% Get the mins/max of the matrix converted to a column-concat vector
redCowMinGrey = min(redCowFileGrey(:));
redCowMaxGrey = max(redCowFileGrey(:));

snapperMinGrey = min(snapperFileGrey(:));
snapperMaxGrey = max(snapperFileGrey(:));

% Display
disp(['<strong> Red Cow min grayscale: ', num2str(redCowMinGrey)]);
```

```
Red Cow min grayscale: 1
```

```
disp(['<strong> Red Cow max grayscale: ', num2str(redCowMaxGrey)]);
```

```
Red Cow max grayscale: 254
```

```
disp(['<strong> Snapper min grayscale: ', num2str(snapperMinGrey)]);
```

```
Snapper min grayscale: 1
```

```
disp(['<strong> Snapper max grayscale: ', num2str(snapperMaxGrey)]);
```

```
Snapper max grayscale: 255
```

Method

For this problem, I used the `rbg2gray` functions to turn both images in grayscale images. This made finding the min and max quite easy. I column concatenated each array, and used the `min`/ `max` functions to find their values.

Results

The results were what I expected them to be. In this case, I think they are completely correct.

Conclusion

Using column concatenation was key here. It made it very easy to look at the entire set of values with one function call. There is not too much else to conclude here.

f: Pout Contrast Adjust

Execute the following commands in Matlab: f = imread('pout.tif'); Calculate and plot the images corresponding to g^.05(x,y)and g^1.5(x,y). Comment on the brightness level and quality of the two images. Note that you may have to convert the image to the double data type (use im2double()) depending on how you choose to solve this

problem. Another thing to note is the maximum and minimum pixel values of the image g. What are min/max values of g before scaling?

```
disp("<strong> Part 1, Section f");
```

Part 1, Section f

```
f = imread('pout.tif');
imshow(f);
```

```
whos f;
```

```
  Name        Size              Bytes  Class    Attributes

  f         291x240             69840  uint8
```

```
% Get min/max before scaling
```

```
minF = min(f(:));
maxF = max(f(:));

disp(['<strong> Before Scaling - min: ', num2str(minF), ' max: ', num2str(maxF)]);
```

Before Scaling - min: 74 max: 224

```
% Convert to doubles so we can raise to powers
f2Double = im2double(f);
whos f2Double;
```

```
  Name            Size              Bytes  Class     Attributes

  f2Double      291x240            558720  double
```

```
% Convert to two clones of doubles.
% Raise values to their appropriate powers element wise
f1 = f2Double.^.5;
f2 = f2Double.^1.5;

% Display new images
disp("<strong> g^.5(x,y)");
```

g^.5(x,y)

```
imshow(f1); % Gets lighter, lower contract
```

```
disp("<strong> g^1.5(x,y)"); % Gets brighter, higher contrast
```

g^1.5(x,y)

```
imshow(f2);
```

Method

For this problem, I used the `im2double` function and element wise multiplication to bring these to their appropriate powers.

Results

The results were what I expected them to be. In this case, I think they are completely correct.

Conclusion

Using g^.5 made the image look a little more dull. Using g^1.5 made the image look quite a bit more sharp.

f2: Alternate Method

You must study the intrans.m function that I provided for you on the class website with this project. I would like you to pay attention to the use of argument checking (use of nargchk, error, strcmp, and isfloat). Explain the use

of “revertclass”. The intent of this exercise is that you develop an understanding of the coding methods used for the function. Use this function as an alternative approach to implement the requested operations for this problem (This is in addition to the solution you provided for (f) above).

```
disp("<strong> Part 1, Section f Alternate Method");
```

Part 1, Section f Alternate Method

```
f3 = intrans(im2double(f), 'stretch',mean2(im2double(f(:))), 1.5);
figure,imshow(f3);
```

```
f4 = intrans(im2double(f), 'stretch',mean2(im2double(f(:))), .5);
figure,imshow(f4);
```

Method

I would be lying if I said this one didn't cause me a little headache. I used the stretch method to multiple the image values. I will say, this made me look very deeply into the `intrans` function and also add the `tofloat` function that was missing. I found that `revertclass` is used to turn the image's data type back to the original input data type after the function was done manipulating it.

Results

The results are not as I expected them to be, and I do not think my results are completely correct. I think my implementation has a bug, but I am not sure what it is. I spent some time evaluating the underlying function and adding log statements within the function, but was not able to come to a valid conclusion of what the problem was. the values of the array in this section and the original section(f), do not line up.

Conclusion

Using g^.5 made the image look a little more dull. Using g^1.5 made the image look quite a bit more cloudy and hard to interpret.

Part 2 (Image Quantization and Scaling)

a: Quantize the grayscale snapper

Quantize the grayscale version of the image snapper to 8 levels and plot it. Explain the method you chose and how it works. Thus if you chose to use built in functions you need to explain how they operate.

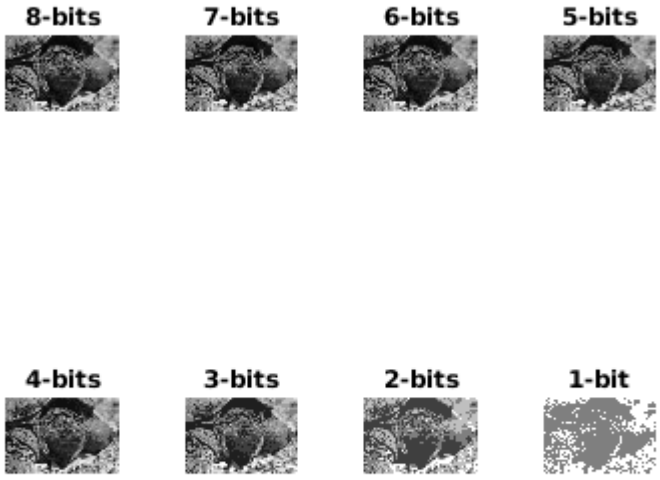
```
% I feel like it would be wasteful to not use the example given to us here.
% I have added detailed notation to demonstrate that I understand the
```

```
% method used. I chose this method over the built in functions because it
% was easier for me to demonstrate understanding.
% In project 0, I actually solved this in a similar way, but used floor's
% instead of ceiling. I think I like using the ceiling better.
disp("<strong> Part 2, Section a");
```

Part 2, Section a

```
snapperFile = imread("snapper.jpg");
% Convert image to grayscale and double
snapperFileDouble = double(rgb2gray(snapperFile));

% Iterate through 8 levels of bits
for numOfBit = 1 : 8
    % 2^8 = 256, 2^7 = 128, etc.
    % More pixels means we have wider range of value levels
    numOfLevel = 2.^ numOfBit;
    % Space between each level
    levelGap = 256 / numOfLevel;
    % Each cell gets divided by the level gap and rounded up the next level
    % above it.
    % ex. if the grayscale value = 65, num bits = 8,level gap = 32
    % 65 / 32 = 2.0312 => which is rounded up to 3.
    % Then when we multiple 3 by the level gap again, we are left with 96.
    % This is appropriate because the range of the level gap is 65-96.
    quantizedImg = uint8((ceil(snapperFileDouble / levelGap) * levelGap) - 1);
    % Fancy subplot for 2 rows, 4 col, and plot number aka starting index
    subplot(2,4,9 - numOfBit), imshow(quantizedImg);
    % Over engineering to get plurality right ;)
    if numOfBit == 1
        name = [num2str(numOfBit) '-bit'];
    else
        name = [num2str(numOfBit) '-bits'];
    end
    title(name);
end
```




```
% Put subplot back.
subplot(1,1,1);
```

Method

I used an implementation similar to the class example. I have added detail notation in the code sample above. In project 0, I actually did something similar to this for scaling, but used `floor` instead of `ceiling`.

Results

The results are as I expected them to be, and I think they are completely correct.

Conclusion

It is clear that for every descending bit, the image is much harder to interpret. It is also clear that the closer the number of bits descends towards 0, the more drastic the image quality loss is. This is because the total % of bits removed is increased as you approach 0.

b: Scale down the image red_cow to 64x43

Scale down the image red_cow to a maximum size of approximately 64 pixels by 43 pixels by appropriately sampling every nth pixel in both the horizontal and vertical directions (you may add extra rows and columns to the image to make it an easier size to work with). Plot your result.

```
disp("<strong> Part 2, Section b");
```

Part 2, Section b

```
redCowFile = imread("red_cow.jpg");
whos redCowFile; % 511x342x3 unint8
```

```
  Name            Size                Bytes  Class    Attributes
```

```
redCowFile      511x342x3           524286  uint8
```

```
% Pull the original row/column sizes from the matrix.
% I am only pulling the z because omitting it will result in column = y*z
% We only column = y here.
[rowSize,columnSize,z] = size(redCowFile); % 511x342x3, as we expected.

rowGap = rowSize / 64;
columnGap = columnSize / 43;

% Write a new array with every nth pixel.
% This is pretty short, but it took me forever.
scaledRedCow = redCowFile(1:ceil(rowGap):511,1:ceil(columnGap):342, :);
imshow(scaledRedCow), title('Red Cow Scaled down');
```


```
whos scaledRedCow %  64x43x3
```

```
  Name              Size             Bytes  Class    Attributes

  scaledRedCow      64x43x3           8256  uint8
```

Method

To scale down the cow image, I created row and column gaps based on the ratio of the original image size to the desired image size. After that, I took row/column gap sized steps through the original image and got 64/43 pixels for our scaled down image.

Results

The results are as I expected them to be, and I think they are completely correct.

Conclusion

The new image has a clear loss in image quality, but is still clearly the same red cow image. There are some rough spots due the nature of how I estimated pixels.

c: Repeat b with imresize

Repeat exercise (b) using the imresize function in Matlab rather than the sub-sampling approach used earlier. You can experiment with different interpolations to see which gives you the most visually pleasing results. Describe your results and why the latter image looks better. After plotting both reduced images side-by-side, use imresize again on both of the reduced image samples (part b and c) but this time - enlarge both images back to their original size using bicubic interpolation. Once again compare the results and comment on what you learned with this exercise.

```
disp("<strong> Part 2, Section c");
```

Part 2, Section c

```
redCowFile = imread("red_cow.jpg");
whos redCowFile; % 511x342x3 unint8
```

```
  Name             Size                Bytes  Class    Attributes

  redCowFile     511x342x3            524286  uint8
```

```
imshow(redCowFile), title('Original');
```

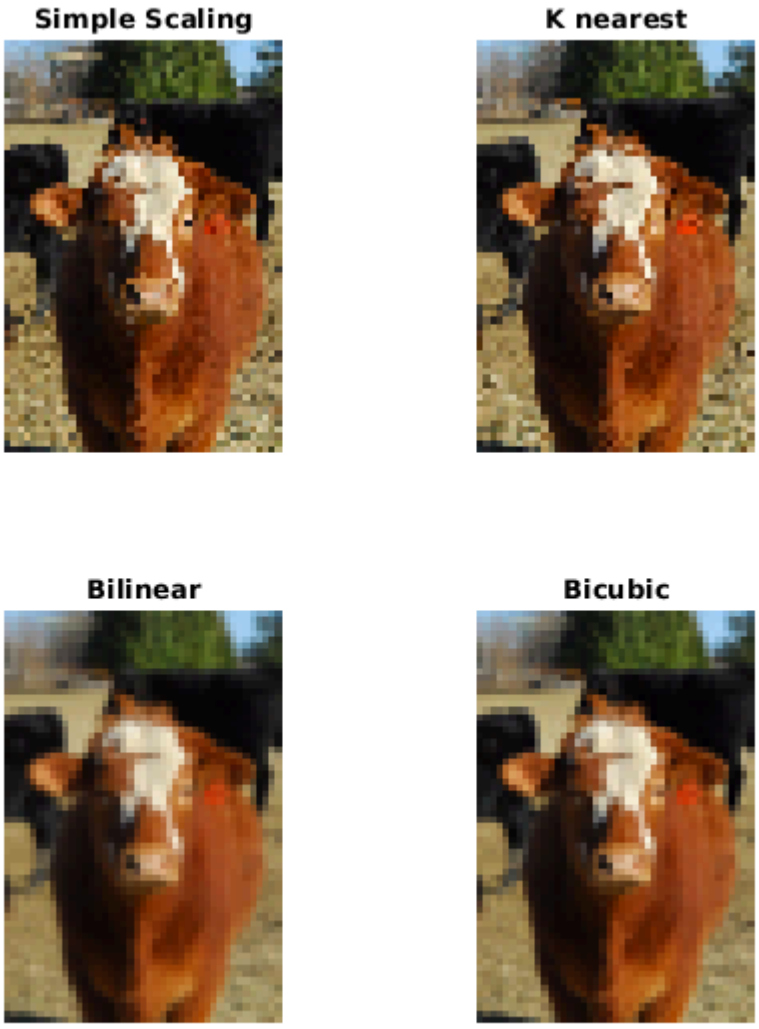

```
% This is pretty short, and it was quick the figure out. I prefer this method to B^^^^.
scaledRedCowNearest = imresize(redCowFile, [64,43], 'nearest');
scaledRedCowBilinear = imresize(redCowFile, [64,43], 'bilinear');
scaledRedCowBicubic = imresize(redCowFile, [64,43], 'bicubic');

% Display all 4 after various scaling.
```

```
disp('<strong> Brought to 64x43 with various methods')
```

Brought to 64x43 with various methods

```
subplot(2,2,1), imshow(scaledRedCow), title('Simple Scaling'); % Variable from section
subplot(2,2,2), imshow(scaledRedCowNearest), title('K nearest');
subplot(2,2,3), imshow(scaledRedCowBilinear), title('Bilinear');
subplot(2,2,4), imshow(scaledRedCowBicubic), title('Bicubic');
```

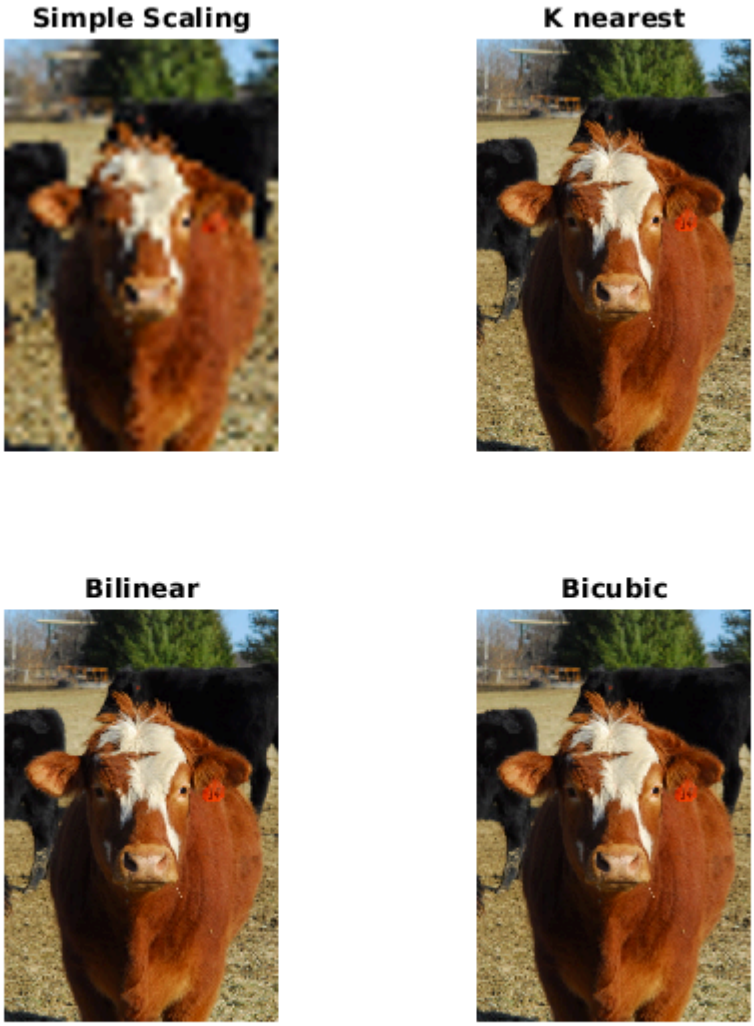



```
subplot(1,1,1);

% Take them back to original size
scaledRedCow = imresize(scaledRedCow, [511,342]);
scaledRedCowNearest = imresize(redCowFile, [511,342], 'nearest');
scaledRedCowBilinear = imresize(redCowFile, [511,342], 'bilinear');
scaledRedCowBicubic = imresize(redCowFile, [511,342], 'bicubic');
disp('<strong> Brought back to 511x342 with various methods')
```

```
Brought back to 511x342 with various methods
```

```
subplot(2,2,1), imshow(scaledRedCow), title('Simple Scaling'); % Variable from section
subplot(2,2,2), imshow(scaledRedCowNearest), title('K nearest');
subplot(2,2,3), imshow(scaledRedCowBilinear), title('Bilinear');
subplot(2,2,4), imshow(scaledRedCowBicubic), title('Bicubic');
```



Method

In addition to what we did in section b, I used the `imresize` function with nearest, bilinear, and bicubic methods for resizing. Afterwards, I used `imresize` with the original dimensions and to bring the images back to scale.

Results

The results are as I expected them to be, and I think they are completely correct.

Conclusion

As one may expected, the simple scaling looks the worst, followed by nearest, folowed by bilinear, and bicubic looked the best. The more input data we used for interpelation, the better our results were.

Part 3 (Geometric Rotations and Affine Homogeneous Transforms)

a: Series of 3d Rotations on Matlab logo

Create a surface plot of the Matlab Logo by using the built-in “membrane” command: L=membrane(1,100); surf(L);

- Explore the Matlab “rotate” command and use it to rotate this surface by -90 degrees (recall positive angles denote CCW rotations) about the x-axis and then -45 degrees about the z-axis (using the fixed axes approach) at which point one ELEE 4920/5920 Project 1 Term II 2020-21 Matlab Logo Surface Plot should be peering approximately into the underside of the membrane plot. You should explore the surface(), surf(), meshgrid and similar commands as you work through this.
- Repeat this exercise using individual rotation matrices (similar to the demos Ex3DRotation.mlx and Affine_test.mlx) to directly manipulate the xyz coordinates and then plot in 3D. Explain your chosen approach (fixed, moving, etc.) and comment on the results relative to the results you received from the Rotate-command work above.

```
disp("<strong> Part 3, Section a");
```

Part 3, Section a

```
subplot(1,1,1);
L1 = membrane(1,100) * 200;
J = surf(L1), title("Rotate -90 on x axis, -45 on z axis");
```

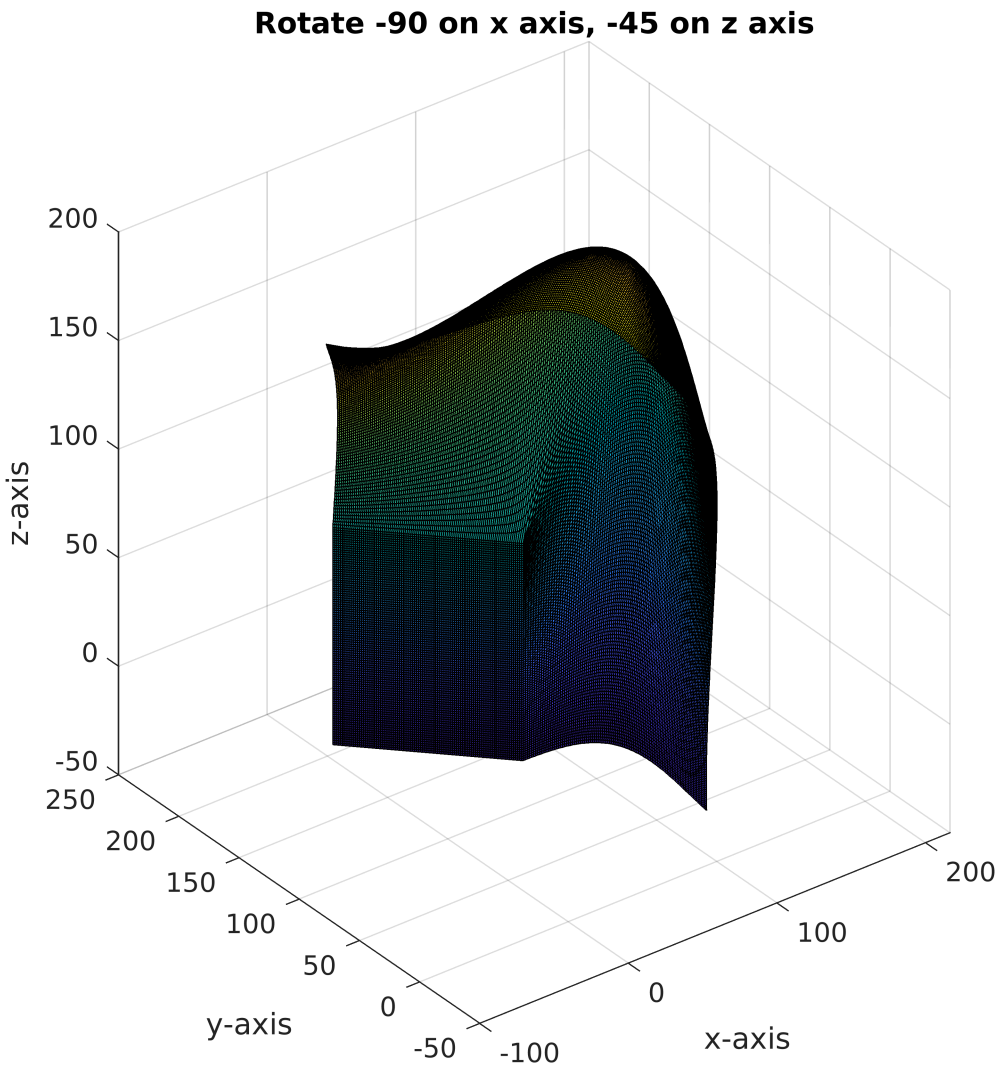
```
J =
  Surface with properties:

      EdgeColor: [0 0 0]
      LineStyle: '-'
      FaceColor: 'flat'
    FaceLighting: 'flat'
      FaceAlpha: 1
          XData: [1×201 double]
          YData: [201×1 double]
          ZData: [201×201 double]
          CData: [201×201 double]

  Show all properties
```

```
xlabel('x-axis')
ylabel('y-axis')
zlabel('z-axis')

rotate(J, [1 0 0], -90);
rotate(J, [0 0 1], -45);
```




```
disp("<strong> Part 3, Section a graduate repeat");
```

Part 3, Section a graduate repeat

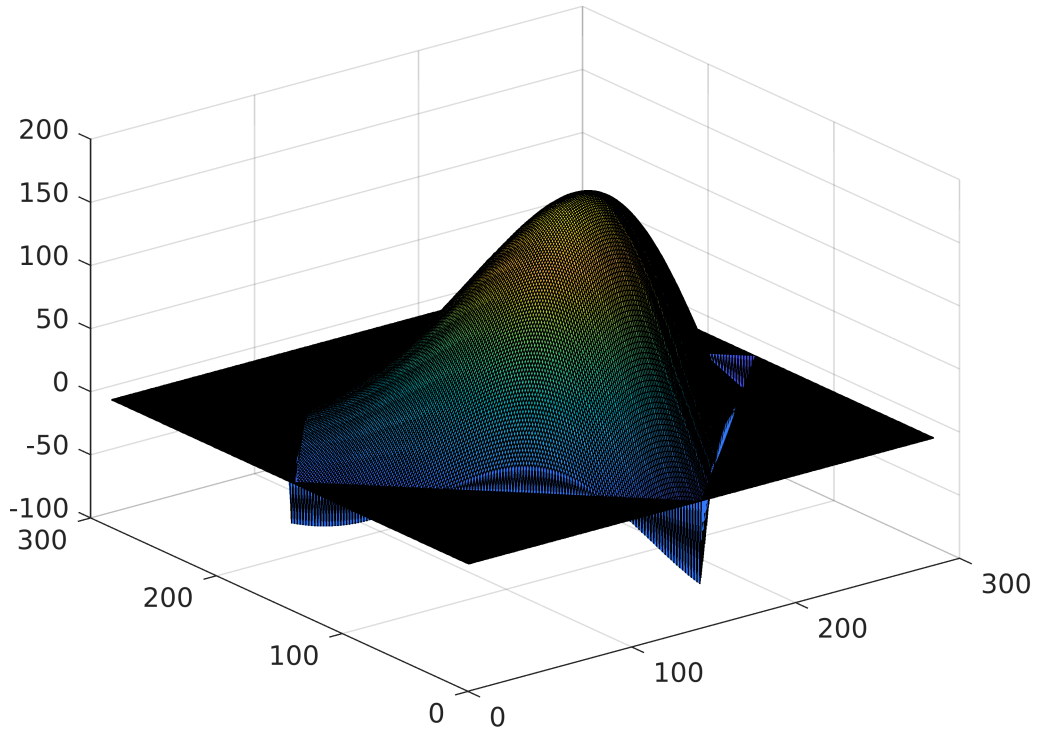
```
% Spent 6+ hours on this and could not get it to work for the membrane.....
% moving on. I was able to reproduce the examples given, but had lots of
% trouble applying it to this context :/
% Tried the rotations from the examples and tried doing affine like in the
% later examples. didn't have much luck.
% Can only get one rotation to work.
figure;
L2 = membrane(1,100) * 200;

Rz = [
    cosd(-45) sind(-45) 0;
    -sind(-45) cosd(-45) 0;
```

```
    0 0 1
    ];
tform_r = affine2d(Rz);
K = imwarp(L2,tform_r);

%% causes syntax errors about the last row needing to be 0 0 1,
% not sure why. ran out of time.
% Rx = [1 0 0;
%     0 cosd(-90) -sind(-90);
%     0 sin(-90) cosd(-90)
%     ]
% tform_r2 = affine2d(Rx);

% K = imwarp(L2, tform_r2)
% K = imwarp()
surf(K);
```



Method

Here, I used the matlab `rotate` command to apply the transformations as requested. Its important to note that the scale of `membrane` function is off for the z axis. I multiplied it by 200 to correct this. The graduate section repeat did not go as smoothly. I spent many hours trying different things and didn't have the best of luck. I was unable to perform multiple rotations and was getting syntax errors I could not debug properly.

Results

The results are not as I expected them to be, and I think they are completely correct for the origina rotation. I do not think they are correct for the graduate repeat. I think that my implementation has a bug.

Conclusion

After performing the `rotate` command, we can see that the turns on the x-axis and z-axis are properly executed because the resulting figure now shows the bottom of the original membrane.

b: Cherry Blossom Afine Games

Now, for the second part of this exercise, you will implement a couple of affine transformations on a Cherry Blossom image (provided with this project). Undergraduates need to first convert this image to a grayscale image whereas Graduate students should process the image in RGB color.

• Use the imwarp() function and the Matlab transform format: [u v] = [x y 1]T for these operations. Consider a review of the useful Matlab Reference on Transformations: Matlab Ref. But again recall, that the transforms given are Transposes of those provided in our notes and the Kay paper because of the different order of operations implemented by Matlab ([x,y,z,1]*T1 in Matlab vs T2*[x,y,z,1]' where T1=T2').

• Load the Cherry.jpg image into the workspace and crop it to a square image (768x768 or 512x512 for example). • Using the affine2D() function and transform matrices you specify, implement a horizontal shear operation on the Cherry image using a value of 0.1.

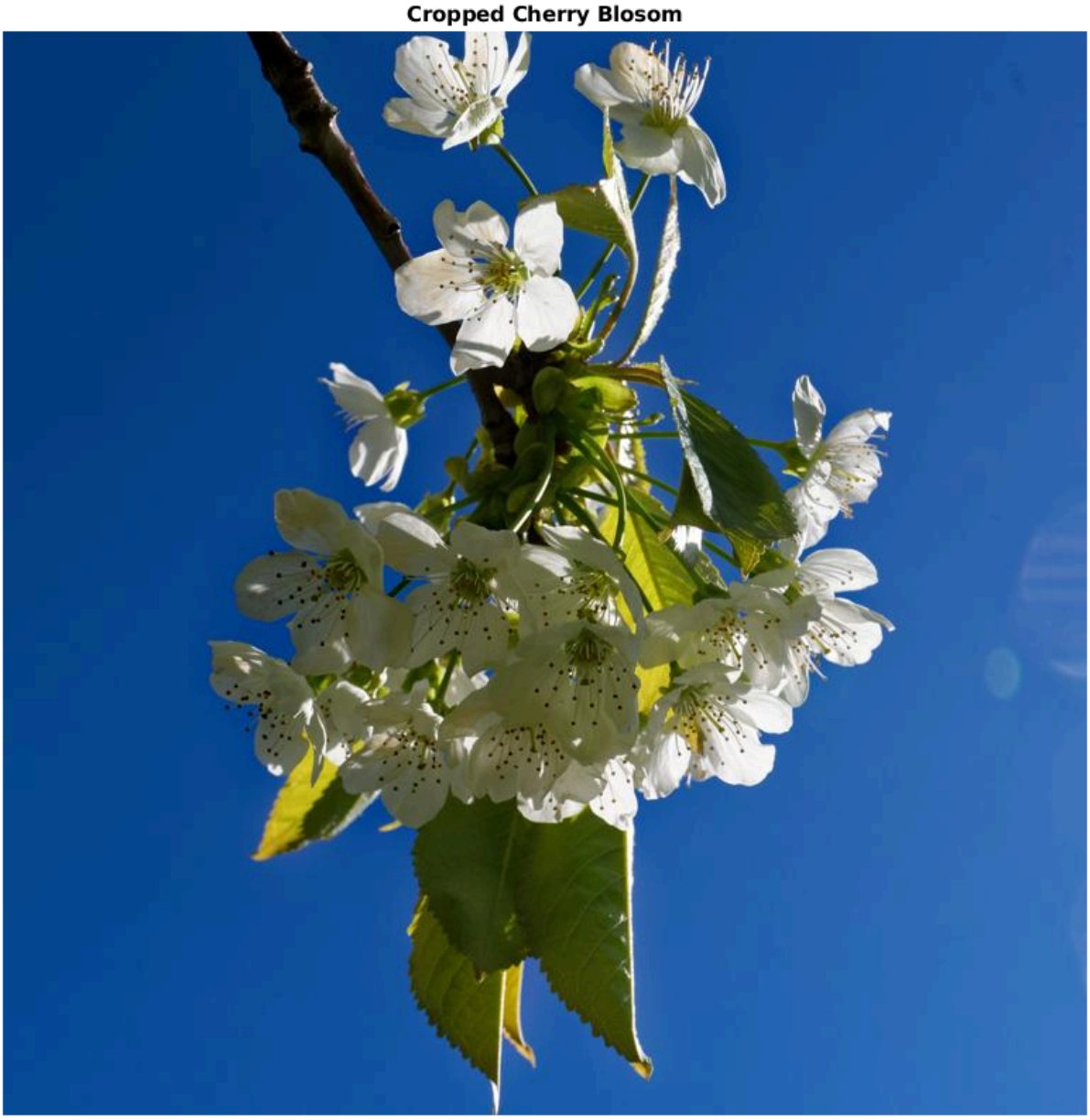
• Using the affine2D() function and transform matrices you specify, implement a combined horizontal (-0.1) and vertical shear (0.2).

• Matlab transform format: [u v] = [x y 1]T and the general approach indicated.

```
disp("<strong> Part 3, Section b");
```

Part 3, Section b

```
cherry = imread("cherry.jpg");
% Crop the image into a  square
r = centerCropWindow2d(size(cherry), [768, 768]);
J = imcrop(cherry,r);
imshow(J), title("Cropped Cherry Blosom");
```

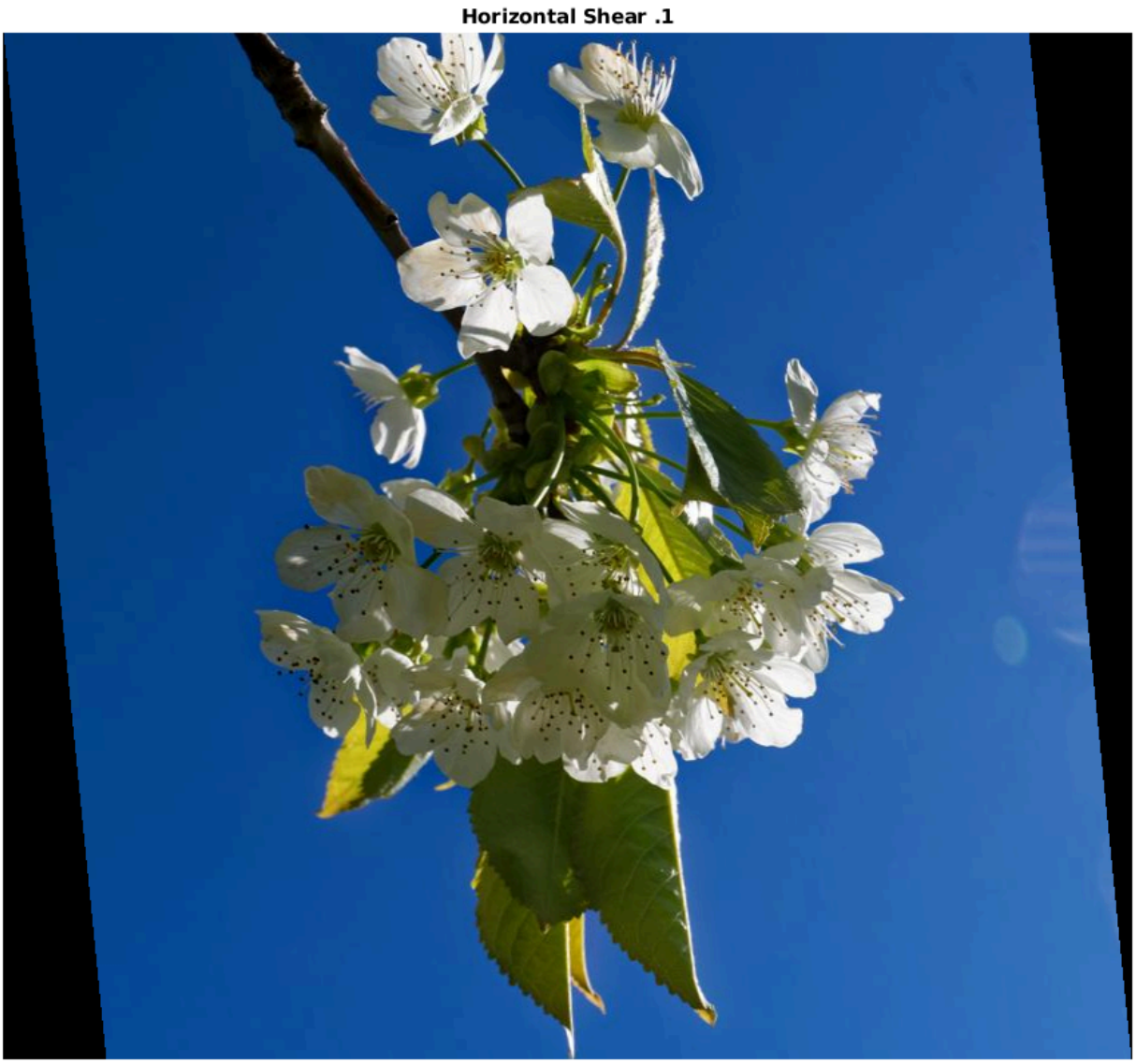



```
% Apply horizontal shear
tform = affine2d([1 0 0; .1 1 0; 0 0 1])
```

```
tform =
  affine2d with properties:

                T: [3×3 double]
    Dimensionality: 2
```

```
K = imwarp(J,tform);
imshow(K), title("Horizontal Shear .1");
```




```
% Apply vertical + horizontal shear
tform = affine2d([1 .2 0; -.1 1 0; 0 0 1])
```

```
tform =
  affine2d with properties:

                T: [3×3 double]
    Dimensionality: 2
```

```
K = imwarp(J,tform);
imshow(K), title("Horizontal Shear -.1. Vertical Shear .2");
```

Method

Here, I used the reference provided and found some nice examples to get started with. You can see the various transforms used above and the values used accordingly. I utilized the `centerCropWindow2d` function to assist in cropping the image. This made things quite a bit easier.

Results

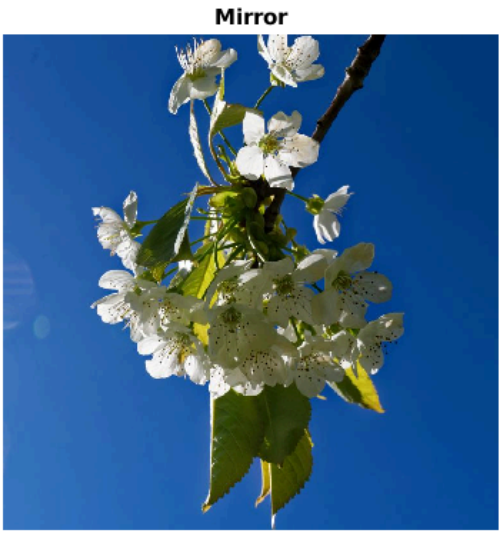
The results are as I expected them to be and I believe they are completely correct.

Conclusion

You can see that using the images have sheared slightly based on the orientation given.

c: Cherry Blossom Afine Mirror

```
% Flip the x-axis to produce a mirror effect.
tform = affine2d([-1 0 0; 0 1 0; 0 0 1]);
K = imwarp(J,tform);
subplot(1,2,1), imshow(J), title("Original");
subplot(1,2,2), imshow(K), title("Mirror");
```



Method

Here, I used the `affine2d` function and negated the x value while keeping the y and z the same. This gave the desired mirror effect becase the x values inverted, while the y and z stayed the same.

Results

The results are as I expected them to be and I believe they are completely correct.

Conclusion

You can see that using the images are mirrors of each other and looking into each other's souls.