Assignment 2: Transforming Images

This assignment is out of a 100 points total. Please turn into Moodle, a .zip file or a .txt file containing a link to a GitHub repository, with the following material:

1. A short report containing answers to all necessary questions and optional questions of your choice. The report should contain **all images** generated by applying your code to images listed in the questions. Additionally, each question in the report should contain names of functions / scripts that you have written for that particular question, and input and output at the MATLAB prompt (as long as the output is small: if the output is an image, do not print it to the MATLAB prompt!). If there is any hand-written part that you will provide me in person, please indicate this in the report.
2. All images generated by applying your code, written out as .jpg or .png files.
3. All scripts and functions written by you for the assignment. Important: make sure the code is commented with a help block in the beginning, and with comments throughout the code.

Optional: any hand-written work that goes along with the assignment can be turned in to me in class the day the assignment is due, slid underneath my door on the due date of the assignment, or scanned in and submitted with the report. Start your assignment soon! Assignment questions begin on Page 2. Happy coding!

On this assignment, we will look at implementing the image transforms discussed in class. All transformed results in this section should be provided on at least the following:

1. The **two** images provided to you
   Image1.png and Image2.png
2. **One** image of your choice.

1 Computing the Inverse Transform Matrix

[20 Points] For several specific transforms, such as scaling, rotation, translation, reflections, and shear, using closed-form expression to compute the inverse of the corresponding transform matrix is faster than using the `inv` function in MATLAB. Also, numerically, the closed-form expression yields more stable results. Here we will obtain the closed-form expressions for the inverses of translation, rotation, reflection, and shear matrices.

1. In class, we saw how to compute the inverse of the scaling matrix, not by a linear algebra approach but by intuiting what it means to scale downwards (shrinking is the multiplicative inverse of enlarging). Similarly, compute the inverse of a translation matrix. Hint: think about what it means to invert translation. We saw that inverting scaling means performing the *opposite* of the particular scaling transform: if you were enlarging the image to get the transformed image, then the image itself is a shrunk version of the transformed image, and vice versa. Apply a similar intuition for translation. (5 points)
2. Using the same intuitions as what you used above, invert a rotation matrix. There is a fundamental relationship between a rotation matrix and its inverse. Can you identify this relationship? (5 points)
3. What should the inverse of a reflection matrix be, and why? (5 points)
4. The inverse for the shear matrix is slightly trickier to intuit, so we will first calculate its inverse using a standard matrix-inversion formula. You will only need the inversion formula for a 2×2 matrix:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \tag{1.1}$$

   Use the above formula to calculate the inverse of the matrix for the shear transform along the x-direction. Intuitively, what does this inverse mean? Can you obtain a similar formula for the inverse of the shear transform along the y-direction? (5 points)

2 Implementing Image Transforms

[80 Points] Use the five-step method discussed in class to implement the image transforms in MATLAB. Your function(s) should have the declaration iimmediately following on the next page.

```
TransformedImage = transformImage(InputImage, TransformMatrix,TransformType);
```

Here, `InputImage` should be a matrix of size $H_{\text{in}} \times W_{\text{in}}$, where H_{in} and W_{in} are the height and width of the input image. Similarly, `TransformedImage` is a matrix of size $H_{\text{out}} \times W_{\text{out}}$, where H_{out} and W_{out} are the height and width of the transformed image. Notice that these are grayscale images. Include the input and transformed images in your report. `TransformMatrix` is a 3×3 matrix that represents a particular transform. `TransformType` is a string, that can take the following values 'scaling', 'rotation', 'translation', 'reflection', 'shear', 'affine', 'homography'. Your function should use `TransformType` to specify the inverse matrix as computed in Section 1. Inverses to Affine and homography transforms can be done using `inv` in MATLAB, but the other inverses **must** be set up using the intuitions from Section 1. Do `help strcmp` to look at comparing strings in MATLAB.

Your function as explained earlier should at the very least take a grayscale image as input and provide a grayscale image as output. This means that if you use the example images provided with the project, you will have to convert those example images to grayscale before you run your code.

You are welcome to write your code so that it uses color images as input and output, but at minimum you should use grayscale images. If you use color images, you need not provide results on grayscale images. If you use color images, note that your transformImage code will have to use a loop that loops over the three channels of the image. Provide results on all images in question (see the top of Section 1) for the following transforms:

1. Change the size of the image to 1080×1920 (make sure the image you start with is not already 1080×1920). (8 points)

2. Reflect the image in the y direction. (8 points)

3. Rotate the image clockwise by 30 degrees. (8 points)

4. Shear the image in the x-direction so that the additional amount added to each x value is 0.5 times each y value. (8 points)

5. Translate the image by 300 in the x-direction and 500 in the y-direction, then rotate the resulting image counterclockwise by 20 degrees, then scale the resulting image down to one-half its size. You should apply the `transformImage` function only once to do this. (16 points)

6. The following two affine transforms: (16 points)

$$\begin{bmatrix} 1 & .4 & .4 \\ .1 & 1 & .3 \\ 0 & 0 & 1 \end{bmatrix} \text{ and } \begin{bmatrix} 2.1 & -.35 & -.1 \\ -.3 & .7 & .3 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

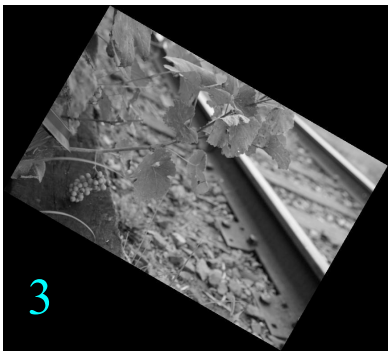
7. The following two homographies: (16 points undergrad, 12 points grad)

$$\begin{bmatrix} .8 & .2 & .3 \\ -.1 & .9 & -.1 \\ .0005 & -.0005 & 1 \end{bmatrix} \text{ and } \begin{bmatrix} 29.25 & 13.95 & 20.25 \\ 4.95 & 35.55 & 9.45 \\ 0.045 & 0.09 & 45 \end{bmatrix} \tag{2.2}$$

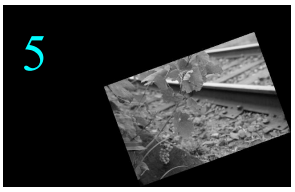
Put the function calls used to perform each of the above transforms into a script: `transforms_script.m`. For Image1.png, you should get the results on the next page (barring overall scale).

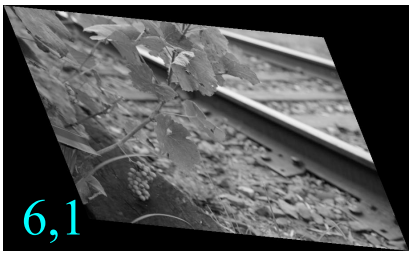

1

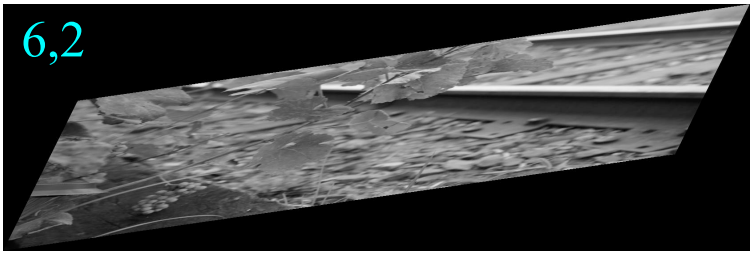

2

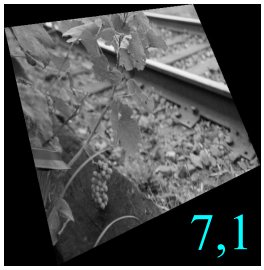

3

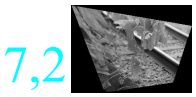

4


5


6,1


6,2


7,1


7,2