Purpose: Become more proficient with using MATLAB for image processing tasks; learn to implement bilinear interpolation in MATLAB. Apply bilinear interpolation as part of a MATLAB program that performs affine transformations of an image.

Procedure: For *each* MATLAB m-file you write, be sure to comment the code sufficiently, and include initial comments as a crude help system. That is, if a user types "help" followed by your m-file name, comments will print on the screen which tell the user what the program does, what the proper syntax is to use the program, and any other useful tidbits. No Image Processing Toolbox programs should be used.

For all images below, the origin should appear in the upper left corner, and the image axes should be such that the x and y dimensions of an individual pixel should appear to be square. The x-axis is for the rows and the y-axis is for the columns. Unless otherwise specified, assume the image has an 8-bit gray level range. The gray levels should map 255 to white and 0 to black.

This project has two main parts. In Part 1, you implement bilinear interpolation for a simple "zooming or shrinking" program. Then, once you are comfortable with bilinear interpolation, you apply it in Part 2 to the application of the affine transform to perform the tasks of rotating and translating an image.

Part 1: Zooming and Shrinking with Bilinear Interpolation

- ➪ Write a MATLAB program capable of zooming and shrinking an image by bilinear interpolation.

  There are many ways to specify "resolution" of an image. For example, the publishing and printing industries often specify image resolution by stating the "dots per inch" or dpi. While this is one way to specify resolution, we're going to do it the more common way for image processing experts. Instead of using dpi in your program, use the desired pixel dimensions of the output image for **both** rows and columns. Note that your MATLAB function should therefore accept *three* inputs: the workspace variable that defines the gray scale input image, and the two separate "size" inputs for the x and y directions (that is, rows and columns, respectively) of the transformed output image (writing the program this way also allows you to intentionally distort the image aspect ratio if you wish). As an example, my version of the program is declared as:

  ```
  function B = ip_bilin_int(A, xnew, ynew)
  ```

  Please use the same order of input arguments in your program that I used in mine. Your MATLAB function should automatically determine the size of the input image.[1]

- ➪ *Before* using your bilinear interpolation program on the high resolution image mentioned below, you should first test your program on a much simpler test image to make sure it performs as expected. The test image I want you to use is the 256 × 256 image of two concentric disks that you created for Project 1. This image is quite simple, and proper operation of your program is therefore easily verified by specifying a new image size; for consistency across project teams, use a new size of 100 × 100. You **must** show in your report your test image (before and after bilinear interpolation, as a figure or figures) and **also** verify ***numerically*** that the interpolated pixel values are correct for a square subimage (at least 3 × 3; larger if needed to prove interpolation occurred) centered at the "southeast" edge of the outer disk in the interpolated image. The relative position of the center of

[1]You may want to investigate the `length()` and/or `size()` commands.

the disks should *not change* as a result of your bilinear interpolation program. Verifying an image processing program on a small, simple test image is *always* a good idea before trying to use it on a larger, more complicated image.

- ➪ The high resolution image you are to use, called `Proj_002_image1.tif`, has the dimensions (given as rows × columns) of 3692×2812. It is a variant of Figure 2.23(a) from your text. Note that many image manipulation programs for the non-expert consumer, such as Adobe Photoshop, may show image dimensions as width × height, which is the opposite of rows × columns. Remember that in the context of image processing, the vertical axis is x and the horizontal axis is y, and the origin is in the upper right hand corner. Furthermore, the publishing or printing industry would likely specify this image's resolution in terms of dpi. Given that the original image from which `Proj_002_image1.tif` was obtained was approximately 2.25 inches wide and 3 inches high, the publishing or printing industry would say this image has a resolution of 1250 dpi. You should start to get used to the wide variety of ways to specify an image; there's no way to avoid it!

  `Proj_002_image1.tif` can be downloaded from the course web site, in the "Images" subdirectory of the "Files" area. Note this is a TIFF (Tagged Image File Format) image.[2] To load a TIF file into MATLAB, you can use the built-in MATLAB routine `imread` to load it into the MATLAB workspace (preferred method), or you can double click on it from the "current directory" tab in MATLAB (but this method provides less control over the variable name) . You can use this technique in the future to load various images into MATLAB, so it's a good thing to learn.

- ➪ As mentioned before, assume `Proj_002_image1.tif` was scanned as a 1250 dpi image. Use your bilinear interpolation program to shrink the image down to 100 dpi; you can calculate what the equivalent new row and column size should be (you may have to round to the nearest integer value, of course). State in your report what this new size is.

- ➪ Next, zoom the previously shrunk image from the previous step back up to the original size. How is this zoomed image different from the original image of `Proj_002_image1.tif`?

- ➪ Now shrink the original image `Proj_002_image1.tif` by different factors in the x and y directions; pick factors of your choice but don't make them ridiculous. This is one simple way to "morph" or distort an image.

Part 2: Affine Transform

- ➪ For this part of the project, download the image `Proj_002_image2.tif` from the course web site, in the "Images" subdirectory of the "Files" area. Note this is also a TIFF image; it is a slight variant of Figure 2.40(a) from your text.

- ➪ Write a MATLAB program to perform image rotation using the appropriate affine transform. Your program should be written in such a way that you can specify any rotation. A suggested example of the m-file function call syntax is:

```
function OUT = ip_rotate(IN, angle)
```

[2]The Tagged Image File Format is a computer file specification for storing raster (i.e., bitmapped) images. TIFF is popular among graphic artists, the publishing/printing industry, and photographers, although JPEG is probably more popular now. TIFF is widely supported by scanning, faxing, word processing, optical character recognition, image manipulation, desktop publishing, and page-layout software.

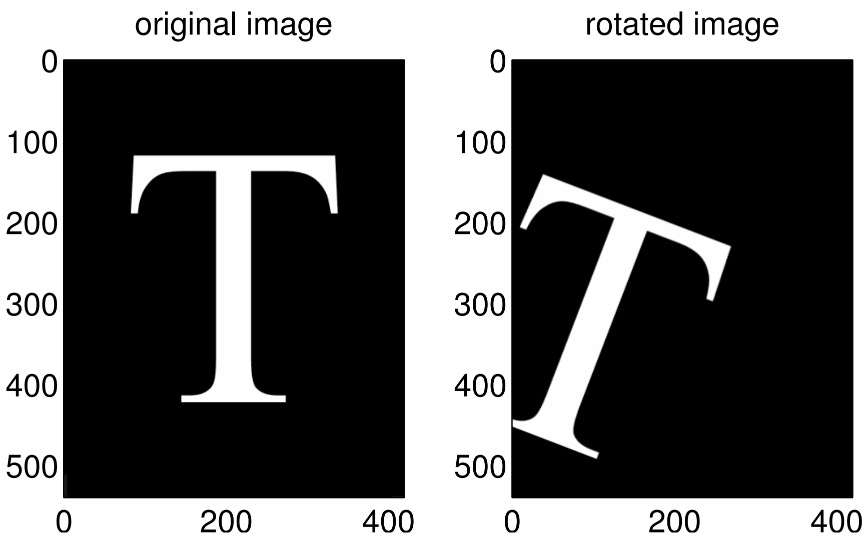

Figure 1: Left: original image `Proj_002_image2.tif`. Right: the figure on the left rotated 21° clockwise, cropped to maintain the same dimensions as the original image. The center point of rotation is the origin of the original image. The inverse mapping approach and bilinear interpolation were used.

- ➪ Rotate the image by 21° clockwise, using the appropriate affine transform.[3] Your program should use the inverse mapping approach and bilinear interpolation, as discussed in the text. Be sure to include in your report the affine matrix **A** that your program determined for this operation, along with the equivalent inverse matrix $\mathbf{A}^{-1}$ that was used for the inverse mapping method.

  When applying geometric spatial transformations to an image, there are two common approaches to determine the dimensions of the output image. One method is to allow the output image to take on whatever maximum dimensions are needed to include all parts of the original image (see Figure 2.41(d) in your text for an example). Another, simpler, method is to constrain the output image to have the same dimensions as the input image, thus possibly "cropping" off parts of the input image that would have ended up "outside" the original image dimensions (see Figure 2.41(c) in your text for an example). To limit your project workload, use the second method: your output image shall have the same dimensions as your input image, so some parts of the input image may appear to be "cropped off."

  Set to a gray level of zero any pixel locations in the output image that have no associated pixel location within the input image (see Figure 1 in this document for an example).

- ➪ Once again, be sure to test your program on a small, simple test image before using the larger image from your textbook! A white square on a black background is an excellent test image for this part of the project. Show your "before and after" rotated test image in your report, in addition to your "before and after" rotated version of `Proj_002_image2.tif`.

- ➪ Once you have written the rotation program, it should now be trivial to write a similar program that performs translation. Translate the image of `Proj_002_image2.tif` in by 150.75 pixels in the x-direction and -67.3 pixels in the y-direction, using the appropriate affine transform. To help you

[3]If you're trying to limp along with the previous edition of the text, Table 2.2 has a typo, and the affine transform method is defined in a way that is the transpose of the method defined in the 4th edition. Consider yourself warned.

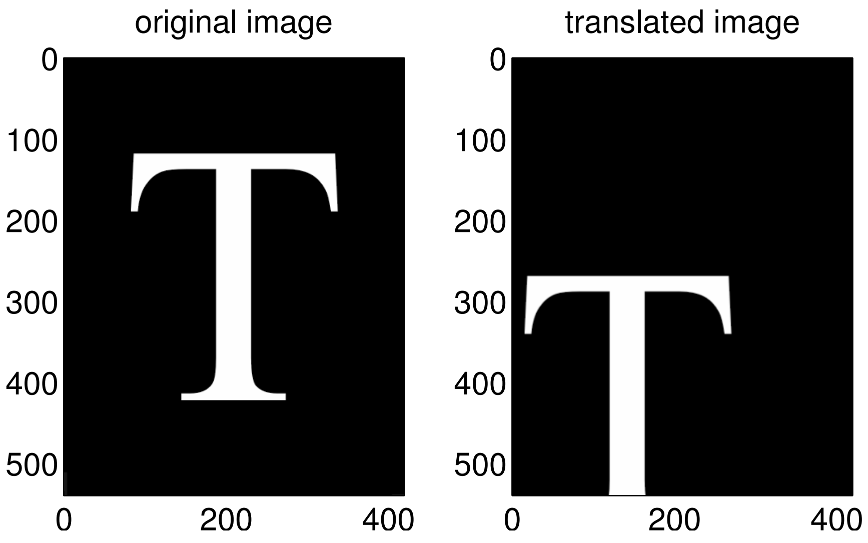

Figure 2: Left: original image `Proj_002_image2.tif`. Right: figure on the left translated by 150.75 pixels in the x-direction and -67.3 pixels in the y-direction, cropped to maintain the same dimensions as the original image. The inverse mapping approach and bilinear interpolation were used.

visualize what this looks like, I've included Figure 2 in this document.[4] Be sure to include in your report the affine matrix $\mathbf{A}$ that your program determined for this operation, along with the equivalent inverse matrix $\mathbf{A}^{-1}$ that was used for the inverse mapping method. *Hint:* some of you may find it easier to write the m-file for translation *before* the m-file for rotation, in that it's a bit easier to visually confirm if it's doing what you expect. But it's up to you in what order you do the tasks for the project. A suggested example of the m-file function call syntax is:

```
function OUT = ip_translate(IN, tx, ty)
```

➪ This isn't a required part of the project, but I hope you can now see the power of using the affine transform. You could now easily write an m-file to scale or shear an image. Or to be even more general, you could write an m-file that accepts just an input image and the transformation matrix $\mathbf{A}$, so that you could combine multiple transformation operations into one pass of the program using a single matrix. One example of an m-file function call syntax is for such a program is:

```
function OUT = ip_affine(IN, A)
```

Once again, you don't need to write this m-file as part of the project. I just wanted to get you thinking about it.

Questions/Discussion: The write-up for this project report should be concise, using wording and formatting suitable for an IEEE technical journal.[5] Be sure to include sufficient figures in your report (with numbered descriptive captions) of any images you create, modify, or that otherwise show results of your

[4]Yes, just like Figure 1, this is very similar to how your result should appear for this part of the project.

[5]Your team's project report must follow the format referenced on the course web site. Be sure to use proper grammar and logical, organized sentences. **Any equations in your report should be typeset with an equation editor** (or use the math mode of LaTeX). All figures and tables should have descriptive numbered captions.

code. If you aren't sure if you should include it as a figure, then you probably should include it! Also, don't make image figures too small just to fit more on a page, and ensure that the font size used for figure titles and axis labels is large enough (similar to the font size in your text). The figures need to be large enough so that a printed version of your report would show the important details of the image.

As a minimum, address these points in your project report.

❶ Answer the questions posed earlier in this document in Part 1: what is the new row and column size of the 100 dpi image, and what are the differences between the original image versus the "shrunken-then-zoomed back to original size" version of the image?

❷ Regarding affine transformations of images, recall that you can perform multiple steps at once by concatenating the sequence of operations into a single affine matrix $\mathbf{A}$. Suppose you wanted to perform these three steps on an image, in this order:[6] 1) scale by a factor of 3 in the x-direction and a factor of 2 in the y-direction, 2) rotate by 30° counterclockwise, and 3) translate by minus 25 pixels in the x-direction and positive 50 pixels in the y-direction. What would be the appropriate single affine matrix $\mathbf{A}$ for this? What would be the equivalent inverse matrix $\mathbf{A}^{-1}$ for the inverse mapping method?

❸ Why does order matter in the item above? Aren't affine transforms just an application of "linear" algebra, and isn't it true that when using linear operations, order doesn't matter? So why does order matter in the item above?

Turn in: For this Project, turn in (as one or more e-mail attachments sent to me as part of a single e-mail message):

➪ An electronic project report (as a PDF file regardless of the program used to create the report). Name your PDF file "`Last1_Last2_proj02.pdf`" please, where "Last1" is the last name of team member 1, and "Last2" is the last name of team member 2. Be sure to follow the format given on the course web site (see the `Admin` section of the course web site).

➪ Any MATLAB m-files you created for this project. There is no need to send me any data or image files that you generated. I'll run your m-files and generate your images myself. If for some reason I need additional files from a particular student, I'll request them separately.

Don't wait until the due date approaches to start this project! It really needs "sink-in" time in your brain (i.e., try it, walk away, come back later and try it again) for it to have the most benefit. Always allow more time than you think you'll need for debugging, and for writing the report.[7] Remember, this is a team effort with a single report per team.

✦✧✦ Have fun... ✦✧✦

[6] Yes, the order matters!

[7] A hastily written report is painfully obvious, unpleasant to read, and will earn only a poor score. Fair warning.