ACIP

Activity 1: Global transforms

Anna, Kimia, Hamza

April 2024

1. First transformation

In this activity, we applied a simple horizontal shearing transformation to an image. We defined this transformation using the matrix T = [1 0 0; -0.3 1 0; 0 0 1]. This matrix tilts the image horizontally, without any vertical displacement or scaling. To implement this transformation, we used MATLAB's **affine2d()** function to create an affine transformation object from T and then applied this object to the image with the **imwarp()** function, effectively altering the image's structure as specified by the matrix.

Figure 1, shows the original image and the transformed image in their corresponding reference frames.

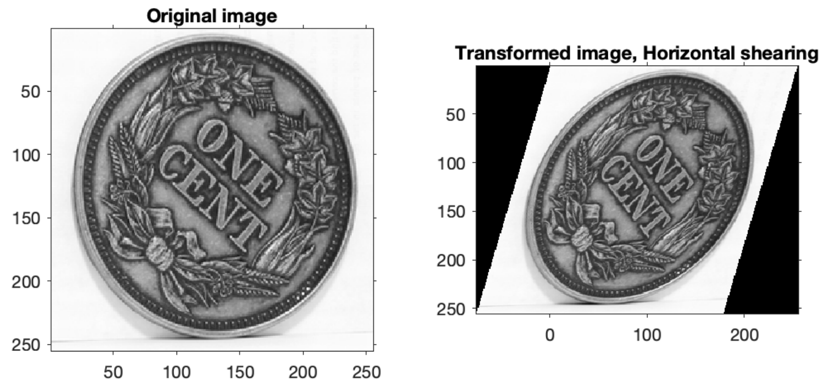

Figure 1: The original image and the transformed image in their corresponding reference frames.

Using the images' reference frames, the effect of the transformation is evident as the limits of the axis have changed; we can see the x-axis in the transformed image is now increased to more than 300. This observation demonstrates the stretching effect that the horizontal shearing has on the image.

However, to ensure a fair comparison between the original and transformed images, we used the **'OutputView'** parameter in the **imwarp** function, to show the images with the same reference frame. By specifying an **imref2d** object, representing the spatial referencing of the original image, we ensure that the transformed image aligns perfectly with the original layout. This helps us to see and compare the specific changes caused by the shearing process.

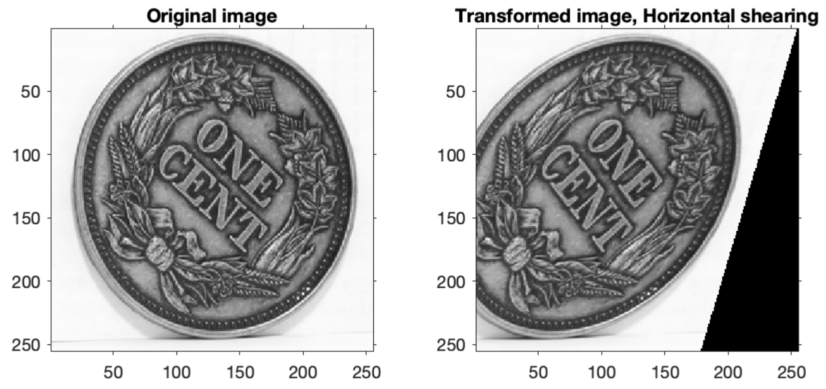

Figure 2: The original image and the transformed image with the same reference frames.

Looking at the transformed image, we notice that it appears slightly less sharp than the original. This loss of sharpness can be attributed to the interpolation process used by the **imwarp** function.

We can break it down into how **imwarp** operates. When applying a specified transformation, **imwarp** computes where each pixel in the input image should map to in the transformed image space; It calculates a new location for each pixel based on the affine transformation matrix. These new positions often result in non-integer coordinates, which do not align perfectly with the original grid of pixel locations.

To resolve this, MATLAB performs interpolation to determine the pixel values at these new, non-integer locations. It resamples these values using various methods such as nearest-neighbor, bilinear, or bicubic interpolation. This process can sometimes reduce the image's sharpness, as the interpolated values may not exactly match the true pixel values.

2. Second transformation

For the second transformation, we chose to implement a rigid transformation, which is a combination of rotation and translation that preserves the shape and size of the object being transformed. The transformation matrix used is defined as T = [cosd(10) sind(10) 0; -sind(10) cosd(10) 0; 10 10 1].

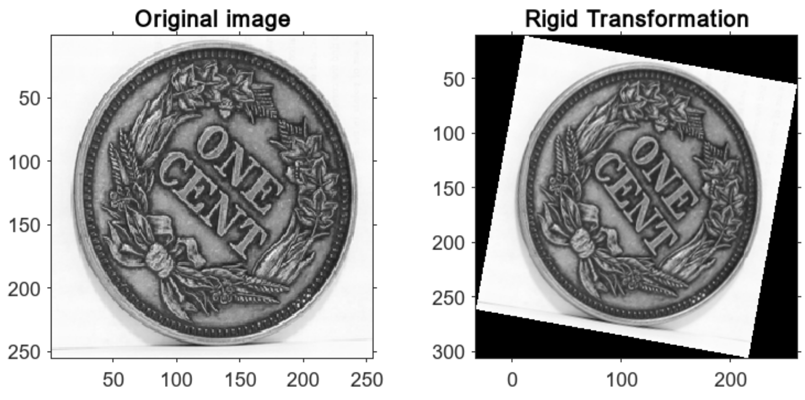

Figure 3: The original image and the transformed image (rigid) in their corresponding reference frames.

This matrix can be broken down into two components: rotation and translation. The elements cosd(10) and sind(10) in the top two rows facilitate the image's rotation by 10 degrees, ensuring that the original dimensions are maintained. The matrix shifts the image 10 units right and 10 units down through the third row [10, 10, 1]. This shift repositions the image within the output frame without altering its orientation.

Following the implementation, we again used the **'OutputView'** parameter in the **imwarp** function to maintain the original reference frame. This consistency allows us to focus solely on the effects of the transformation.

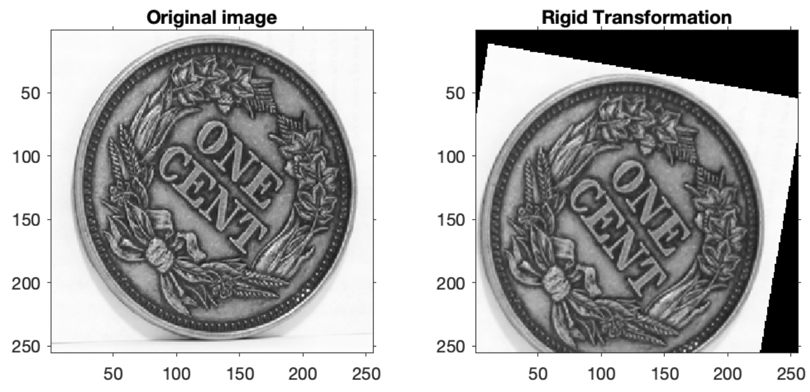

Figure 4: The original image and the transformed image (rigid) with the same reference frames.

Like the first transformation, the sharpness of the transformed image is slightly reduced. This phenomenon is due to the interpolation required to estimate pixel values at new locations in the transformed image space. The interpolation process, while essential for filling in pixel values at non-integer coordinates, can sometimes smooth out details, especially in areas of high contrast or with sharp edges.