HW1 An An Yu

- Implementation:

  - Rotate the 4 corners of I_1 (the original image). The height and width spanned by the resulting 4 points are the minimal dimensions of I_2 (the rotated image).

  - p_f is the rotated p_0 expressed in the new frame. The new frame is defined as follows (assuming x-axis points down and y-axis points right): x-axis is the vertical line that passes through the leftmost point in the rotated image, and y-axis is the horizontal line that passes through the topmost point in the rotated image.

    $$p_f = -min(p_{cr}),\ p_{cr} = R(\theta)(p_c - p_o) \quad (1)$$

  - For every pixel in I_2, calculate its corresponding coordinates in I_1 by reverse mapping :

    $$p_{I1} = p_0 + R(-\theta)(p_{I2} - p_f) \quad (2)$$

    Find the four neighbors of p_{I1} and do bilinear interpolation on them, set the resulting value to p_{I2}

- Compare with 'imwarp':

Bilinear interpolation produces fuzzier edges in the image. Taking a weighted average of pixel values from 4 neighbors is similar to apply an size 2x2 averaging/blurring window on the original image, only difference is the weight is defined as the relative euclidean distance from each neighbor.

figure 1. rotate_image(60^o, (128,128))

figure 2. imwarp(60^o,'linear')

- p_0 doesn't affect the result of our rotated image because p_f always cancels its effect, aligning with the x and y axes.

  $p_{I1} = p_0 + R(-\theta)(p_{I2} - p_f)$ where $p_f = -min(R(\theta)(p_c - p_o))$

  $\Rightarrow \quad p_{I1} = p_0 + R(-\theta)p_{I2} + R(-\theta)min(R(\theta)(p_c)) - R(-\theta)R(\theta)p_0$

  $\Rightarrow \quad p_{I1} = R(-\theta)p_{I2} + R(-\theta)min(R(\theta)(p_c))$ (independent of p_0)

  The first arrowhead holds because min is taken over p_c, which is independent of p_o.

- Results:

figure 3. rotate_image, po = (0,0)

figure 4. rotate_image, po = (40,100)

figure 5. rotate_image, po = (128,128)

figure(6).
rotate_image(135, (40,100))

figure(7).
rotate_image(225, (128,128))