Ethan Hartman
Assignment 3

Please note, through all steps, I generally would display the output image to myself so I could understand what the computer was analyzing, to tweak many of the threshold values.

Noise removal Techniques and Methods:

For removing noise, I just used a 9x9 gaussian blur with a standard deviation of 0 along with thresholding the image. The blur was partially for removing small unwanted objects such as dust, and to reduce light within the dice. My dot and dice calculating methods were intelligent enough to weed out any excess noise which seeped through.

Image Threshold Criteria:

I started off with the max threshold being at 255 because we wanted to keep the darkest objects, such as the dots on the dice. For the minimum, I started with a lower number, but this was prone to leaving slight white on the brighter dots. I had to keep raising the minimum threshold until I was at the sweet spot, where dots would not touch because the minimum was so high, and glares were mostly removed.

Separating Foreground from Background:

For separating the foreground from the background, I just had to use my threshold technique. Because the cloth was always black, it turned completely black after thresholding, while dice were made pure white.

Extracting Dice from the Image:

Ethan Hartman
Assignment 3

To extract dice from the images, I would use the thresholder image and locate the contours with cv2.findContours. I then would iterate through all contours, convert them into 4 corners with cv2.approxPolyDP. After this, I could calculate the approximate offset of the corners to find the x, and y offset, then the magnitude of the two offsets for the estimated height and width of the die. Next, if the height was larger than a specified minimum dice size, which I extracted from trial and error, I would estimate the number of horizontal and vertical dice touching each other, based on the width, height, and pre-defined max dice size. I would then use a nested for loop to calculate the horizontal and vertical dice, calculating their corners with vector calculations, and on calculating corners, I would wrap them in a minimum area rectangle to represent the corners of the dice, then append them to a valid dice corner list.

Counting Dice Dots:

To calculate the number of dots within each dice, I iterated through the valid dice corner list, created a mask with only the pixels within the corners of the die, then found the contours within that dice using our cv2.findContours method. After finding the potential contours, I iterated through all of them to validate their shapes, and size. I tested to ensure that their arc length was within a predetermined min and max size, and it was a certain circularity, to be considered a circle. If it passed this test, I would increase the dot counter for that dice by one. After all contours have been accounted for in the current dice, if the dot count was between 1-6, we have a valid dice, so we'd increase the counter for the dice with that dot count, and the total dot sum as well. If it was not between 1-6, I would count this as an unknown, so I'd increase that count by 1.