def determine\_checkerboard\_size(image\_path):

    # Read the image

    img = cv2.imread(image\_path, cv2.IMREAD\_COLOR)

    # Convert to grayscale for processing

    gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

    # Circle detection and removal

    circles = cv2.HoughCircles(gray, cv2.HOUGH\_GRADIENT, dp=1, minDist=30, param1=50, param2=30, minRadius=5, maxRadius=40)

    if circles is not None:

        circles = np.uint16(np.around(circles))

        for i in circles[0,:]:

            cv2.circle(gray, (i[0], i[1]), i[2], (127,127,127), -1)  # Overwrite circle with average checkerboard color

    # Image enhancement using adaptive histogram equalization

    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))

    enhanced\_img = clahe.apply(gray)

    # Thresholding

    \_, threshed = cv2.threshold(enhanced\_img, 127, 255, cv2.THRESH\_BINARY\_INV)

    # Finding contours

    contours, \_ = cv2.findContours(threshed, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

    # Assuming the largest contour (after a certain size) is the checkerboard

    max\_contour = max(contours, key=cv2.contourArea)

    # Applying perspective transform if the contour has 4 points (approximation)

    if len(max\_contour) == 4:

        pts1 = np.array([max\_contour[0], max\_contour[1], max\_contour[2], max\_contour[3]], dtype="float32")

        side = max(img.shape)

        pts2 = np.array([[0, 0], [side - 1, 0], [side - 1, side - 1], [0, side - 1]], dtype="float32")

        matrix = cv2.getPerspectiveTransform(pts1, pts2)

        img = cv2.warpPerspective(img, matrix, (side, side))

    # Check for 8x8 checkerboard

    pattern\_size = (7, 7)

    found, \_ = cv2.findChessboardCorners(img, pattern\_size)

    if found:

        return "8 x 8 (British/American rules)"

    # Check for 10x10 checkerboard

    pattern\_size = (9, 9)

    found, \_ = cv2.findChessboardCorners(img, pattern\_size)

    if found:

        return "10 x 10 (International rules)"

    return "Checkerboard not recognized"