Function Documentation: `extract_radius_mask`

1 Description

The `extract_radius_mask` function processes an image to identify and mask circles representing dishes and weights. It extracts circles, filters out duplicates, sorts them, and applies transformations to create a final image with masked dishes and weights.

2 Function Definition

```
def extract_radius_mask(filepath, save=None):
    frame = cv2.imread(filepath)
    modified_frame = apply_curve_transformation(frame, piecewise_linear_transformation,
    ↪  0.00, 0.70)
    gray = cv2.cvtColor(modified_frame, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (9, 9), 2)

    dish_circles = cv2.HoughCircles(blurred, cv2.HOUGH_GRADIENT, dp=1, minDist=300,
                               param1=20, param2=29, minRadius=1150, maxRadius=1350)
    unique_indices = eliminate_duplicate_circles(dish_circles[0], center_threshold=300)
    dish_circles = np.round(dish_circles[0, unique_indices]).astype("int")

    # sort the dish circles left to right first, then top to bottom
    height, width, channels = frame.shape
    left_dish_circles = dish_circles[np.where(dish_circles[:,0] < 6*width/10)]
    right_dish_circles = dish_circles[np.where(dish_circles[:,0] >= 6*width/10)]
    left_dish_circles = left_dish_circles[np.argsort(left_dish_circles[:,1])]
    right_dish_circles = right_dish_circles[np.argsort(right_dish_circles[:,1])]
    dish_circles = np.concatenate((left_dish_circles, right_dish_circles), axis=0)

    # Draw the filled dish circle on the mask as white (keeping)
    mask = np.zeros_like(frame)
    for (x, y, r) in dish_circles:
        cv2.circle(mask, (x, y), r, (255, 255, 255), -1)

    # Mask the image before searching for the weights
    masked_frame = cv2.bitwise_and(modified_frame, mask)
    blurred = cv2.cvtColor(masked_frame, cv2.COLOR_BGR2GRAY)

    weight_circles = cv2.HoughCircles(blurred, cv2.HOUGH_GRADIENT_ALT, dp=1, minDist=1,
                               param1=60, param2=0.30, minRadius=90, maxRadius=140)[0]

    # sort by radius here
    radii_indices = np.argsort(weight_circles[:,2])[::-1]
    weight_circles = weight_circles[radii_indices]
```

```
    unique_indices = eliminate_duplicate_circles(weight_circles, center_threshold=180)
    weight_circles = np.round(weight_circles[unique_indices]).astype("int")

    # Draw the filled weight circle on the mask as black (discard)
    for (x, y, r) in weight_circles:
        in_center = False
        for (x2, y2, r2) in dish_circles:
            if np.sqrt((x-x2)**2+(y-y2)**2) < 800:
                in_center = True
                break
        if in_center == False:
            cv2.circle(mask, (x, y), r+20, (0, 0, 0), -1)

    # Now for each dish we calculate necessary rotation to place the
    # three masses on the right (centered at 0 radians) and the single
    # mass on the left (centered at pi radians)
    rot_angles = []
    for i, (x1, y1, r2) in enumerate(dish_circles):
        left_weights_x = []
        left_weights_y = []
        right_weights_x = []
        right_weights_y = []
        for j, (x2, y2, r2) in enumerate(weight_circles):
            if np.sqrt((x1-x2)**2+(y1-y2)**2) < 800:
                continue
            if (x1-x2)**2+(y1-y2)**2 <= 1250*1250:
                if x2 < x1:
                    left_weights_x.append(x2)
                    left_weights_y.append(y2)
                else:
                    right_weights_x.append(x2)
                    right_weights_y.append(y2)
        if len(left_weights_x) > 0 and len(right_weights_x) > 0:
            xl, yl = np.average(left_weights_x), np.average(left_weights_y)
            xr, yr = np.average(right_weights_x), np.average(right_weights_y)
            theta = np.arctan2(yr-y1,xr-x1)
            if len(left_weights_x) > len(right_weights_x):
                theta += np.pi
            rot_angles.append(theta)
        else:
            rot_angles.append(0)

    # optionally output the masked first frame
    if save is not None:
        final_image = np.zeros((height, width * 2, channels), dtype=np.uint8)

        final_image[0:height, 0:width] = frame
        masked_frame = cv2.bitwise_and(frame, mask)
        final_image[0:height, width:2*width] = masked_frame
        for i, (x, y, r) in enumerate(dish_circles):
            # extract the single dish
```

```
            dish = masked_frame[y - r:y + r, x - r:x + r]
            # rotate it
            rotation_matrix = cv2.getRotationMatrix2D((int(r),int(r)),
            ↪  rot_angles[i]*180/np.pi, 1)
            rotated_dish = cv2.warpAffine(dish, rotation_matrix, (2*int(r), 2*int(r)))
            # re-insert it
            final_image[y - r:y + r, x - r + width:x + r + width] = rotated_dish
            cv2.putText(final_image, f'd{i+1:02}', (x, y), cv2.FONT_HERSHEY_SIMPLEX, 4,
            ↪  (255, 255, 255), 2)
            cv2.putText(final_image, f'r{rot_angles[i]*180/np.pi:5.2}', (x, y+100),
            ↪  cv2.FONT_HERSHEY_SIMPLEX, 4, (255, 255, 255), 2)
        cv2.imwrite(save, final_image)

    return mask, dish_circles, rot_angles
```

3 Function Explanation

3.1 Step-by-Step Breakdown

Function 1: Read and Transform Image

Read the image from the specified file path and apply a curve transformation to it.

```
frame = cv2.imread(filepath)
modified_frame = apply_curve_transformation(frame, piecewise_linear_transformation, 0.00,
↪  0.70)
```

Explanation: The function reads the image from the specified `filepath` and applies a curve transformation to it using `apply_curve_transformation`. The transformed image is stored in `modified_frame`.

Function 2: Convert and Blur Image

Convert the transformed image to grayscale and apply Gaussian blur.

```
gray = cv2.cvtColor(modified_frame, cv2.COLOR_BGR2GRAY)
blurred = cv2.GaussianBlur(gray, (9, 9), 2)
```

Explanation: The image is converted to grayscale and then blurred using Gaussian blur to prepare it for circle detection.

Function 3: Detect and Sort Dish Circles

Detect dish circles using the Hough Circle Transform, filter duplicates, and sort them.

```
dish_circles = cv2.HoughCircles(blurred, cv2.HOUGH_GRADIENT, dp=1, minDist=300,
                                param1=20, param2=29, minRadius=1150, maxRadius=1350)
unique_indices = eliminate_duplicate_circles(dish_circles[0], center_threshold=300)
dish_circles = np.round(dish_circles[0, unique_indices]).astype("int")

height, width, channels = frame.shape
left_dish_circles = dish_circles[np.where(dish_circles[:,0] < 6*width/10)]
right_dish_circles = dish_circles[np.where(dish_circles[:,0] >= 6*width/10)]
left_dish_circles = left_dish_circles[np.argsort(left_dish_circles[:,1])]
right_dish_circles = right_dish_circles[np.argsort(right_dish_circles[:,1])]
dish_circles = np.concatenate((left_dish_circles, right_dish_circles), axis=0)
```

Explanation: Detect circles representing dishes, filter out duplicates, and sort them first by their x-coordinate and then by y-coordinate.

Function 4: Create Dish Mask

Create a mask for the detected dish circles and apply it to the modified image.

```
mask = np.zeros_like(frame)
for (x, y, r) in dish_circles:
    cv2.circle(mask, (x, y), r, (255, 255, 255), -1)

masked_frame = cv2.bitwise_and(modified_frame, mask)
blurred = cv2.cvtColor(masked_frame, cv2.COLOR_BGR2GRAY)
```

Explanation: Create a binary mask for the dish circles and apply it to the image. This mask is used to isolate the dish regions for weight detection.

Function 5: Detect Weights and Sort

Detect weights using the Hough Circle Transform, sort them by radius, and filter duplicates.

```
weight_circles = cv2.HoughCircles(blurred, cv2.HOUGH_GRADIENT_ALT, dp=1, minDist=1,
                                param1=60, param2=0.30, minRadius=90, maxRadius=140)[0]

radii_indices = np.argsort(weight_circles[:,2])[::-1]
weight_circles = weight_circles[radii_indices]

unique_indices = eliminate_duplicate_circles(weight_circles, center_threshold=180)
weight_circles = np.round(weight_circles[unique_indices]).astype("int")
```

Explanation: Detect circles representing weights, sort them by their radius, and filter out duplicates.

Function 6: Update Mask for Weights

Update the mask to discard weight circles that overlap with dish circles.

```
for (x, y, r) in weight_circles:
    in_center = False
    for (x2, y2, r2) in dish_circles:
        if np.sqrt((x-x2)**2+(y-y2)**2) < 800:
            in_center = True
            break
    if in_center == False:
        cv2.circle(mask, (x, y), r+20, (0, 0, 0), -1)
```

Explanation: Update the mask to exclude weight circles that are close to dish circles by drawing them in black.

Function 7: Calculate Rotation Angles

Calculate the rotation angles for each dish to align weights correctly.

```
rot_angles = []
for i, (x1, y1, r2) in enumerate(dish_circles):
    left_weights_x = []
    left_weights_y = []
    right_weights_x = []
    right_weights_y = []
    for j, (x2, y2, r2) in enumerate(weight_circles):
        if np.sqrt((x1-x2)**2+(y1-y2)**2) < 800:
            continue
        if (x1-x2)**2+(y1-y2)**2 <= 1250*1250:
            if x2 < x1:
                left_weights_x.append(x2)
                left_weights_y.append(y2)
            else:
                right_weights_x.append(x2)
                right_weights_y.append(y2)
    if len(left_weights_x) > 0 and len(right_weights_x) > 0:
        xl, yl = np.average(left_weights_x), np.average(left_weights_y)
        xr, yr = np.average(right_weights_x), np.average(right_weights_y)
        theta = np.arctan2(yr-y1,xr-x1)
        if len(left_weights_x) > len(right_weights_x):
            theta += np.pi
        rot_angles.append(theta)
    else:
        rot_angles.append(0)
```

Explanation: Calculate the rotation angles needed for each dish to align weights properly based on their positions.

Function 8: Save and Return Results

Optionally save the final image and return the mask, dish circles, and rotation angles.

```
if save is not None:
    final_image = np.zeros((height, width * 2, channels), dtype=np.uint8)

    final_image[0:height, 0:width] = frame
    masked_frame = cv2.bitwise_and(frame, mask)
    final_image[0:height, width:2*width] = masked_frame
    for i, (x, y, r) in enumerate(dish_circles):
        dish = masked_frame[y - r:y + r, x - r:x + r]
        rotation_matrix = cv2.getRotationMatrix2D((int(r),int(r)),
        ↪  rot_angles[i]*180/np.pi, 1)
        rotated_dish = cv2.warpAffine(dish, rotation_matrix, (2*int(r), 2*int(r)))
        final_image[y - r:y + r, x - r + width:x + r + width] = rotated_dish
        cv2.putText(final_image, f'd{i+1:02}', (x, y), cv2.FONT_HERSHEY_SIMPLEX, 4, (255,
        ↪  255, 255), 2)
        cv2.putText(final_image, f'r{rot_angles[i]*180/np.pi:5.2}', (x, y+100),
        ↪  cv2.FONT_HERSHEY_SIMPLEX, 4, (255, 255, 255), 2)
    cv2.imwrite(save, final_image)

return mask, dish_circles, rot_angles
```

Explanation: If a `save` path is provided, save the final image showing both the original and masked images. The function returns the mask, dish circles, and rotation angles.