实验四

1. 加载图像

```
ef load_image(image_path):
    """加载图像并转换为 RGB 格式"""
    img = cv2.imread(image_path)
    if img is None:
        raise FileNotFoundError(f"Image not found: {image_path}")
    return cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

2. 图像预处理

```
preprocess_image(gray):
    """图像预处理"""
    # 高斯模糊降噪
    blurred = cv2.GaussianBlur(gray, (5, 5), 1.4)

    # 自适应直方图均衡化
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    equalized = clahe.apply(blurred)

    return equalized
```

3. 边缘检测

```
def detect_edges(gray, low=None, high=None):
    """边缘检测"""
    # 自动计算 Canny 阈值
    if low is None or high is None:
        median = np.median(gray)
        sigma = 0.33
        low = int(max(0, (1.0 - sigma) * median))
        high = int(min(255, (1.0 + sigma) * median))

    edges = cv2.Canny(gray, low, high)

    # 形态学操作增强边缘
    kernel = np.ones((3,3), np.uint8)
    edges = cv2.dilate(edges, kernel, iterations=1)
    edges = cv2.erode(edges, kernel, iterations=1)

    return edges
```

4. 霍夫直线检测

```
def detect_lines(edges, params):
    """霍夫直线检测"""
    lines = cv2.HoughLinesP(edges, **params)

    # 如果没有检测到直线，尝试更宽松的参数
    if lines is None or len(lines) == 0:
        relaxed_params = params.copy()
        relaxed_params['threshold'] = max(10, params['threshold'] // 2)
        relaxed_params['minLineLength'] = max(10,
params['minLineLength'] // 2)
        lines = cv2.HoughLinesP(edges, **relaxed_params)

    return lines
def filter_lines(lines, min_angle_diff=15, max_angle_diff=165):
    """过滤和合并相似的直线"""
    if lines is None or len(lines) == 0:
        return None

    # 计算每条直线的角度和长度
    line_info = []
    for line in lines:
        x1, y1, x2, y2 = line[0]
        angle = np.degrees(np.arctan2(y2-y1, x2-x1)) % 180
        length = np.sqrt((x2-x1)**2 + (y2-y1)**2)
        line_info.append({'line': line, 'angle': angle, 'length':
length})

    # 按长度排序
    line_info.sort(key=lambda x: -x['length'])

    filtered_lines = []
    angle_groups = []

    for info in line_info:
        line, angle, length = info['line'], info['angle'],
info['length']

        # 忽略接近水平或垂直的直线（可根据需要调整）
        if min_angle_diff < angle < max_angle_diff:
            # 检查是否与已存在的直线角度相似
            similar = False
            for group in angle_groups:
```

```
                if abs(group['angle'] - angle) < 10:  # 角度差小于10度视
为相似
                    similar = True
                    # 检查是否共线
                    x1, y1, x2, y2 = line[0]
                    gx1, gy1, gx2, gy2 = group['line'][0]
                    # 简单的共线性检查
                    d1 = abs((y2-y1)*gx1 - (x2-x1)*gy1 + x2*y1 - y2*x1)
/ np.sqrt((y2-y1)**2 + (x2-x1)**2)
                    d2 = abs((y2-y1)*gx2 - (x2-x1)*gy2 + x2*y1 - y2*x1)
/ np.sqrt((y2-y1)**2 + (x2-x1)**2)
                    if d1 < 10 and d2 < 10:  # 距离阈值
                        # 合并直线（取端点最远的两个点）
                        all_points = np.array([line[0],
group['line'][0]]).reshape(-1,2)
                        hull = cv2.convexHull(all_points)
                        if len(hull) >= 2:
                            new_line = np.array([[hull[0][0][0],
hull[0][0][1], hull[-1][0][0], hull[-1][0][1]]])
                            group['line'] = new_line
                            group['length'] = np.sqrt((new_line[0][2]-
new_line[0][0])**2 + (new_line[0][3]-new_line[0][1])**2)
                            similar = True
                            break

            if not similar:
                filtered_lines.append(line)
                angle_groups.append({'line': line, 'angle': angle})

    return np.array(filtered_lines) if filtered_lines else None
```

5. 霍夫圆检测

```
def detect_circles(gray, params):
    """霍夫圆检测"""
    circles = cv2.HoughCircles(gray, **params)
    return np.uint16(np.around(circles)) if circles is not None else
None
```

6. 绘制检测结果

```
result_img = img_rgb.copy()
    # 绘制直线
    if lines is not None:
        for line in lines:
            x1, y1, x2, y2 = line[0]
            cv2.line(result_img, (x1,y1), (x2,y2), (255,0,0), 2)
    # 绘制圆形
    if circles is not None:
        for circle in circles[0,:]:
            cv2.circle(result_img, (circle[0],circle[1]), circle[2],
(0,255,0), 2)
            cv2.circle(result_img, (circle[0],circle[1]), 2, (0,0,255),
3)
    return result_img
```

7. 结果分析

```
def analyze_results(lines, circles):
    """结果分析"""
    analysis = {}
    # 圆参数分析
    if circles is not None:
        largest_circle = max(circles[0,:], key=lambda x:x[2])
        analysis['circle'] = {
            'center': (int(largest_circle[0]), int(largest_circle[1])),
            'diameter': int(largest_circle[2]*2)
        }
    # 直线角度分析
    if lines is not None:
        longest_line = max(lines, key=lambda x: np.linalg.norm(x[0][:2]-
x[0][2:]))
        x1, y1, x2, y2 = longest_line[0]
        angle = np.degrees(np.arctan2(y2-y1, x2-x1)) % 180
        analysis['weld'] = {
            'points': [(int(x1), int(y1)), (int(x2), int(y2))],
            'angle': float(f"{abs(angle):.2f}")
        }
    return analysis
```

8. 主函数

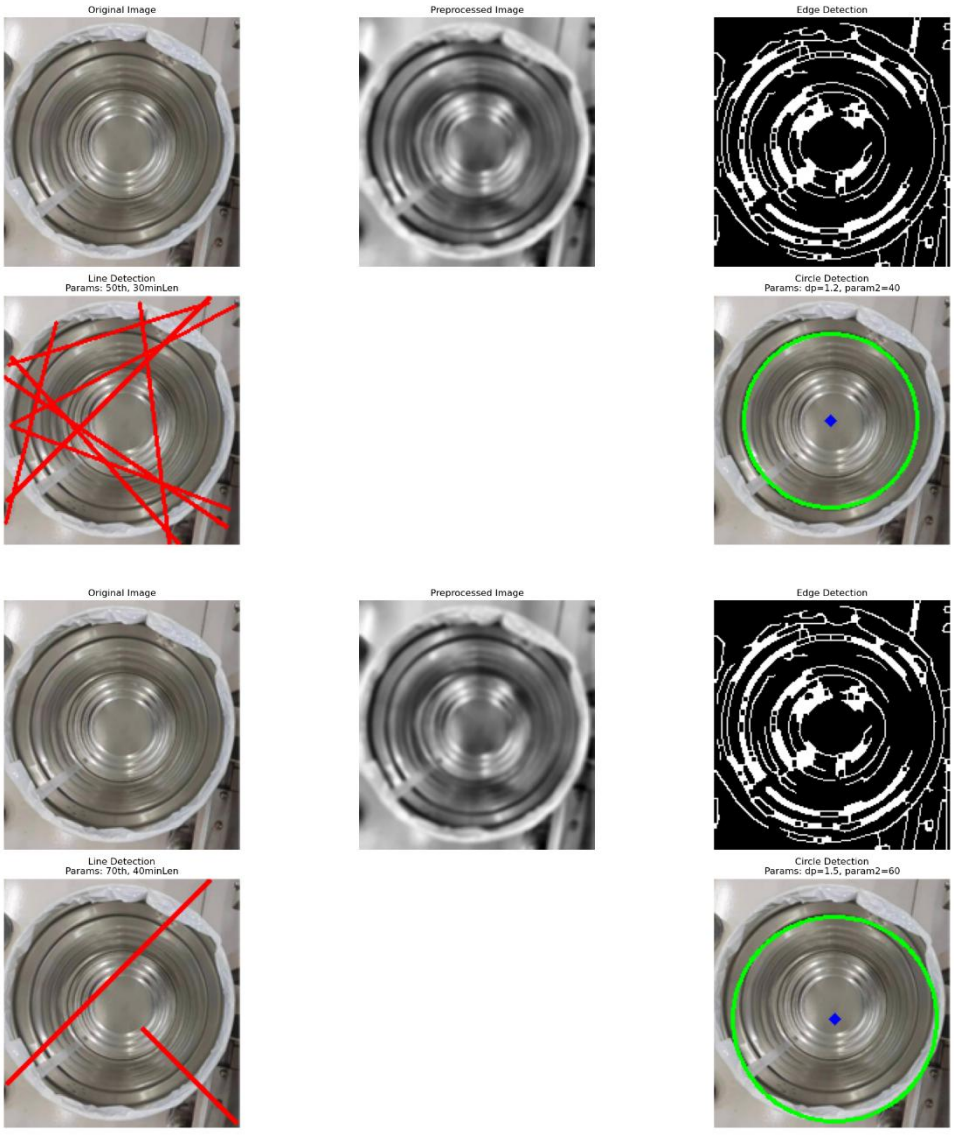
```
f __name__ == "__main__":
    # 参数配置
```

```python
IMAGE_PATH = "D:\Samples/bucket4.png"

# 改进的霍夫参数
LINE_PARAMS = [
    {   # 默认参数
        'rho': 1,
        'theta': np.pi/180,
        'threshold': 50,  # 降低阈值以检测更多直线
        'minLineLength': 30,  # 减少最小长度
        'maxLineGap': 20  # 增加最大间隙
    }
]

CIRCLE_PARAMS = [
    {   # 默认参数
        'method': cv2.HOUGH_GRADIENT,
        'dp': 1.2,
        'minDist': 50,
        'param1': 200,
        'param2': 40,
        'minRadius': 20,
        'maxRadius': 100
    }
]

try:
    # 加载图像
    img_rgb = load_image(IMAGE_PATH)
    gray = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2GRAY)

    # 改进的图像预处理
    processed = preprocess_image(gray)
    edges = detect_edges(processed)

    # 创建可视化画布
    plt.figure(figsize=(20, 10))

    # 显示原始图像和边缘检测
    plt.subplot(2, 3, 1)
    plt.imshow(img_rgb)
    plt.title("Original Image")
    plt.axis('off')

    plt.subplot(2, 3, 2)
```

```
        plt.imshow(processed, cmap='gray')
        plt.title("Preprocessed Image")
        plt.axis('off')

        plt.subplot(2, 3, 3)
        plt.imshow(edges, cmap='gray')
        plt.title("Edge Detection")
        plt.axis('off')

        # 直线检测
        for i, params in enumerate(LINE_PARAMS, 4):
            lines = detect_lines(edges, params)
            filtered_lines = filter_lines(lines)
            result_img = draw_detections(img_rgb, lines=filtered_lines)
            plt.subplot(2, 3, i)
            plt.imshow(result_img)
            plt.title(f"Line Detection\nParams: {params['threshold']}th,
{params['minLineLength']}minLen")
            plt.axis('off')

        # 圆检测
        for i, params in enumerate(CIRCLE_PARAMS, 6):
            circles = detect_circles(processed, params)
            result_img = draw_detections(img_rgb, circles=circles)
            plt.subplot(2, 3, i)
            plt.imshow(result_img)
            plt.title(f"Circle Detection\nParams: dp={params['dp']},
param2={params['param2']}")
            plt.axis('off')

        plt.tight_layout()
        plt.savefig('improved_hough_detection.jpg')
        plt.show()

        # 使用最佳参数进行分析
        best_lines = detect_lines(edges, LINE_PARAMS[0])
        filtered_lines = filter_lines(best_lines)
        best_circles = detect_circles(processed, CIRCLE_PARAMS[0])
        analysis = analyze_results(filtered_lines, best_circles)

        # 英文结果输出
        print("\n=== Final Analysis ===")
        if 'circle' in analysis:
            c = analysis['circle']
```

```
            print(f"Barrel Bottom - Center: {c['center']}, Diameter:
{c['diameter']}px")
        if 'weld' in analysis:
            w = analysis['weld']
            print(f"Weld Seam - Endpoints: {w['points']}, Angle:
{w['angle']}°")

    except Exception as e:
        print(f"Error: {str(e)}")
```

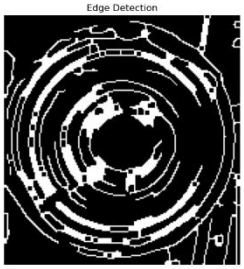
9. 不同参数下执行结果



Original Image

Preprocessed Image

Edge Detection



Line Detection
Params: 40th, 50minLen

Circle Detection
Params: dp=1.6, param2=50