**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)

1. **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

1. **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

1. **霍夫直线检测**

ef 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

1. **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

1. **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

1. **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

1. **主函数**

f \_\_name\_\_ == "\_\_main\_\_":

    # 参数配置

    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)}")

1. **不同参数下执行结果**

图片包含 电子, 照相机, 光盘

AI 生成的内容可能不正确。图片包含 电子, 照相机

AI 生成的内容可能不正确。图片包含 电子, 照相机, 光盘

AI 生成的内容可能不正确。