Extreme Detailed Analysis of Fiber Optic Image Processing Code

Import Statements

`import cv2`

- **Purpose**: Imports OpenCV (Open Source Computer Vision Library)
- **Usage**: Provides image processing functions like:
  - Image loading/saving
  - Color space conversions
  - Morphological operations
  - Contour detection
  - Circle fitting algorithms

`import numpy as np`

- **Purpose**: Imports NumPy library with alias 'np'
- **Usage**: Provides:
  - Array operations
  - Mathematical functions
  - Boolean indexing for masks
  - Coordinate manipulation

Function 1: `apply_filter(image)`

Purpose

Converts image to binary (black/white) and applies morphological operations to clean noise

Detailed Breakdown:

`result = image.copy()`

- Creates a deep copy of the input image
- Prevents modifying the original image
- Allocates new memory for the result

`if len(result.shape) == 3:`

- Checks if image has 3 dimensions
- Shape format: (height, width, channels)
- 3 dimensions = color image (BGR)
- 2 dimensions = grayscale image

```
result = cv2.cvtColor(result, cv2.COLOR_BGR2GRAY)
```

- Converts BGR color image to grayscale
- Formula: Gray = 0.299*R* + *0.587*G + 0.114*B
- Reduces 3 channels to 1 channel
- Only executes if image is colored

```
_, result = cv2.threshold(result, 127, 255, cv2.THRESH_BINARY)
```

- Applies binary threshold
- Parameters:
  - `result`: Input grayscale image
  - `127`: Threshold value (middle of 0-255 range)
  - `255`: Maximum value assigned to pixels above threshold
  - `cv2.THRESH_BINARY`: Binary threshold type
- Operation:
  - If pixel > 127: set to 255 (white)
  - If pixel ≤ 127: set to 0 (black)
- `_` ignores the returned threshold value

```
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
```

- Creates elliptical structuring element
- Size: 5x5 pixels
- Shape: Circular/elliptical pattern
- Used for morphological operations
- Better than rectangular for preserving circular features

```
result = cv2.morphologyEx(result, cv2.MORPH_CLOSE, kernel)
```

- Performs morphological closing operation

- Closing = Dilation followed by Erosion
- Effects:
  - Fills small holes in white regions
  - Connects nearby white regions
  - Smooths boundaries
  - Preserves overall shape
- Helps clean up noise in binary image

`return result`

- Returns the processed binary image

Function 2: `find_annulus_and_inner_circle(filtered_image)`

Purpose

Identifies the black ring (annulus) and white center region, then calculates their circular boundaries

Detailed Breakdown:

`contours, _ = cv2.findContours(filtered_image, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)`

- Finds all contours in binary image
- Parameters:
  - `filtered_image`: Binary input image
  - `cv2.RETR_TREE`: Retrieves all contours with hierarchy
  - `cv2.CHAIN_APPROX_SIMPLE`: Compresses horizontal/vertical segments
- Returns:
  - `contours`: List of contour points
  - `_`: Hierarchy information (ignored here)

`black_mask = np.zeros_like(filtered_image)`

- Creates empty mask same size as input
- All pixels initialized to 0 (black)
- Will store annulus pixels

`inner_white_mask = np.zeros_like(filtered_image)`

- Creates empty mask for inner white region
- Same size as input image
- Will store center white pixels

`black_pixels = (filtered_image == 0)`

- Creates boolean array
- True where pixels are black (value 0)
- False where pixels are white (value 255)

`num_labels, labels = cv2.connectedComponents(filtered_image)`

- Finds connected white regions
- Each connected region gets unique label
- Returns:
  - `num_labels`: Total number of regions found
  - `labels`: Array with region labels

Variable Initialization

```python
inner_circle_info = None
outer_circle_info = None
annulus_contour = None
```

- Initializes storage for circle parameters
- Will store center coordinates and radii

`black_contours, _ = cv2.findContours((~filtered_image).astype(np.uint8), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)`

- `~filtered_image`: Inverts image (white→black, black→white)
- `.astype(np.uint8)`: Converts boolean to 8-bit image
- `cv2.RETR_EXTERNAL`: Only retrieves outermost contours
- Finds contours of black regions

Main Loop: `for contour in black_contours:`

Processes each black region to find the annulus

`temp_mask = np.zeros_like(filtered_image)`

- Creates temporary mask for current contour

`cv2.drawContours(temp_mask, [contour], -1, 255, -1)`

- Draws filled contour on mask
- Parameters:
  - `[contour]`: Single contour in list
  - `-1`: Draw all contours (only one here)
  - `255`: Color (white)
  - `-1`: Fill contour (not just outline)

`x, y, w, h = cv2.boundingRect(contour)`

- Gets rectangular bounding box
- Returns:
  - `x, y`: Top-left corner coordinates
  - `w, h`: Width and height

`filled_mask = np.zeros_like(filtered_image)`

- Creates mask for filled contour region

`cv2.drawContours(filled_mask, [contour], -1, 255, -1)`

- Fills the contour completely
- Used to check what's inside

`inner_white = (filtered_image == 255) & (filled_mask == 255)`

- Boolean AND operation
- True only where:
  - Original image is white (255)
  - AND inside the filled contour
- Identifies white pixels inside black region

`if np.any(inner_white):`

- Checks if any white pixels exist inside
- If true, this black region is an annulus

Inside the if block:

- `black_mask = black_mask | (temp_mask & black_pixels)`
  - OR operation adds annulus pixels to mask
  - AND ensures only actual black pixels added
- `inner_white_mask = inner_white_mask | inner_white`
  - Adds inner white pixels to mask
- Stores largest annulus contour by area

Circle Fitting Section

`if annulus_contour is not None:`

Executes if annulus was found

Outer Circle Calculation:

```python
full_mask = np.zeros_like(filtered_image)
cv2.drawContours(full_mask, [annulus_contour], -1, 255, -1)
```

- Creates mask of entire annulus region

```python
outer_points = np.column_stack(np.where(full_mask > 0))
```

- `np.where(full_mask > 0)`: Finds all non-zero pixels
- Returns tuple of (row_indices, col_indices)
- `np.column_stack`: Combines into (y,x) coordinates

```python
outer_points_cv = outer_points[:, [1, 0]].astype(np.float32)
```

- Swaps columns: (y,x) → (x,y) for OpenCV
- Converts to float32 for circle fitting

```python
(outer_x, outer_y), outer_radius = cv2.minEnclosingCircle(outer_points_cv)
```

- Finds minimum enclosing circle
- Returns center (x,y) and radius

Inner Circle Calculation:

Similar process for inner white pixels:

- Extracts white pixel coordinates
- Converts to OpenCV format
- Fits minimum enclosing circle

Return Statement

```python
return black_mask, inner_white_mask, inner_circle_info, outer_circle_info
```

Returns all calculated data

Function 3: `calculate_concentricity(inner_circle_info, outer_circle_info)`

Purpose

Measures how well-centered the inner circle is within the outer circle

Detailed Breakdown:

`if inner_circle_info is None or outer_circle_info is None:`

- Validates both circles exist
- Returns None if either missing

Distance Calculation:

```python
center_distance = np.sqrt((inner_center[0] - outer_center[0])**2 +
                          (inner_center[1] - outer_center[1])**2)
```

- Euclidean distance formula
- Calculates pixel distance between centers
- sqrt((x2-x1)^2 + (y2-y1)^2)

Results Dictionary:

- `center_offset`: Absolute distance in pixels
- `normalized_offset`: Distance divided by outer radius (0-1 scale)
- Stores both center coordinates

Function 4: `segment_original_image(original, black_mask, inner_white_mask)`

Purpose

Separates original image into different regions based on masks

Detailed Breakdown:

`black_mask = black_mask.astype(bool)`

- Converts mask to boolean type
- Ensures proper logical operations

`combined_mask = black_mask | inner_white_mask`

- OR operation combines both masks
- True where either mask is true

Cleaned Image:

```python
cleaned_image = original.copy()
cleaned_image[~combined_mask] = 0
```

- Keeps only annulus and inner regions
- Sets everything else to black (0)
- `~` inverts the mask

White Region Image:

python

```
white_region_image = original.copy()
white_region_image[~inner_white_mask] = 0
```

- Keeps only inner white region
- Everything else becomes black

Black Region Image:

python

```
black_region_image = original.copy()
black_region_image[~black_mask] = 0
```

- Keeps only annulus region
- Everything else becomes black

Outside Region Image:

python

```
outside_region_image = original.copy()
outside_region_image[combined_mask] = 0
```

- Keeps only regions outside annulus
- Annulus and inner become black

Function 5: `create_visualization_image(...)`

Purpose

Creates annotated image showing detected circles and measurements

Detailed Breakdown:

Image Preparation:

python

```
overlay = original.copy()
if len(overlay.shape) == 2:
    overlay = cv2.cvtColor(overlay, cv2.COLOR_GRAY2BGR)
```

- Ensures image is in color (BGR) format
- Converts grayscale to color if needed

Color Coding Masks:

```python
overlay_vis[black_mask] = [0, 255, 0]  # Green
overlay_vis[inner_white_mask] = [255, 0, 0]  # Red
```

- Uses boolean indexing
- BGR format: [Blue, Green, Red]

Drawing Inner Circle:

```python
center = (int(inner_circle_info['center'][0]), int(inner_circle_info['center'][1]))
radius = int(inner_circle_info['radius'])
cv2.circle(overlay_vis, center, radius, [255, 255, 0], 2)
cv2.circle(overlay_vis, center, 3, [255, 255, 0], -1)
```

- Converts float coordinates to integers
- Draws circle outline (thickness=2)
- Draws filled center point (radius=3)
- Yellow color: [255, 255, 0]

Drawing Connection Line:

```python
cv2.line(overlay_vis, inner_center, outer_center, [255, 0, 255], 2)
```

- Draws line between centers
- Magenta color: [255, 0, 255]
- Shows concentricity offset

Adding Text:

```python
text = f"Offset: {concentricity_info['center_offset']:.2f}px"
cv2.putText(overlay_vis, text, (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
            1, [255, 255, 255], 2, cv2.LINE_AA)
```

- Formats offset to 2 decimal places
- Position: (10, 30) pixels from top-left
- White text with anti-aliasing

Function 6: `main()`

Purpose

Orchestrates entire processing pipeline

Detailed Breakdown:

Image Loading:

```python
image_path = r"C:\Users\Saem1001\Documents\GitHub\IPPS\App\images\img63.jpg"
original_image = cv2.imread(image_path)
```

- Raw string (r"") preserves backslashes
- `cv2.imread` loads image as NumPy array
- Returns None if file not found

Error Checking:

```python
if original_image is None:
    print(f"Error: Could not load image from {image_path}")
    return
```

- Validates image loaded successfully
- Exits early if failed

Processing Pipeline:

1. Apply filter (binarization + morphology)

2. Find annulus and inner circle
3. Calculate concentricity
4. Extract pixel positions
5. Segment original image
6. Create visualization

Position Extraction:

```python
black_positions = np.column_stack(np.where(black_mask))
```

- `np.where` returns (row, col) indices
- `column_stack` creates Nx2 array
- Each row is [y, x] coordinate

Console Output:

Prints detailed measurements:

- Pixel counts for each region
- Circle parameters (center, radius, diameter)
- Concentricity metrics
- Quality assessment based on offset

File Saving Operations:

Binary Masks:

```python
cv2.imwrite("black_mask.png", black_mask.astype(np.uint8) * 255)
```

- Converts boolean to 0/255 values
- Saves as grayscale PNG

Text Files:

```python
np.savetxt("black_pixel_positions.txt", black_positions, fmt='%d')
```

- Saves coordinates as integers
- One coordinate pair per line

Measurement Report:

```python
with open("circle_measurements.txt", "w") as f:
```

- Creates formatted text report
- Uses context manager for file handling
- Includes all measurements and quality rating

Return Dictionary:

Returns comprehensive results:

- All images (original, filtered, masks, segments)
- Pixel position arrays
- Circle measurements
- Concentricity data

Main Execution Block

```python
if __name__ == "__main__":
    results = main()
```

- Only runs if script executed directly
- Not when imported as module
- Stores results in variable

Data Flow Summary

1. **Input**: Color/grayscale fiber optic image
2. **Filtering**: Convert to binary, clean with morphology

3. **Detection**: Find black annulus with white center
4. **Measurement**: Fit circles, calculate concentricity
5. **Segmentation**: Separate image regions
6. **Visualization**: Annotate with circles and measurements
7. **Output**: 9 images, 3 text files, 1 measurement report

Key Computer Vision Concepts Used

- **Thresholding**: Separates light/dark regions
- **Morphological Operations**: Cleans binary images
- **Connected Components**: Identifies separate regions
- **Contour Analysis**: Finds object boundaries
- **Circle Fitting**: Estimates best-fit circles
- **Boolean Masking**: Selects specific pixels
- **Coordinate Transformations**: Converts between formats