DIP Assignment 3

Fahad Kamran

November 19, 2023

Question 1: Image Comparison

0.1 Steps:

- Import the OpenCV library.
- Create a list of image file paths (`images`).
- Use nested loops to iterate through each pair of images.
- Read the images (`img1` and `img2`) using OpenCV's `imread` function.
- Calculate the absolute difference (`diff`) between the two images.
- Convert the difference image to grayscale (`mask`).
- Apply a binary threshold to identify color variations.
- Convert the thresholded image back to BGR color space (`mask`).
- Use bitwise AND to highlight color variations (`highlighted_diff`).
- Resize the resulting image to 30% of its original size.
- Display the resulting image using OpenCV's `imshow`.
- Wait for a key press (`waitKey`) and close the display window (`destroyAllWindows`).

0.2 Results

Figure 1: Input Image 1

Figure 2: Input Image 2

Figure 3: Input Image 3

Figure 4: Input Image 4

Figure 5: Output Image 1

Figure 6: Output Image 2

Figure 7: Output Image 3

Figure 8: Output Image 4

Figure 9: Output Image 5

Figure 10: Output Image 6

Question 2: Distance Calculation

0.3 Steps:

- Import the OpenCV and NumPy libraries.
- Define a function (`detect_red_dots`) to detect red dots in an image using color thresholding in the HSV color space.
- Define a function (`calculate_distance`) to calculate the Euclidean distance between two points.
- Read an image (`image`) using OpenCV's `imread` function.
- Detect red dots in the image using the `detect_red_dots` function.
- Draw rectangles around the detected red dots using OpenCV's `rectangle` function.
- If there are at least 4 red dots, calculate distances between pairs of red dots using the `calculate_distance` function.
- Print the calculated distances.
- Resize the image before displaying it to 60% of its original size.
- Display the image with drawn contours using OpenCV's `imshow`.
- Wait for a key press (`waitKey`) and close the display window (`destroyAllWindows`).

0.4 Results

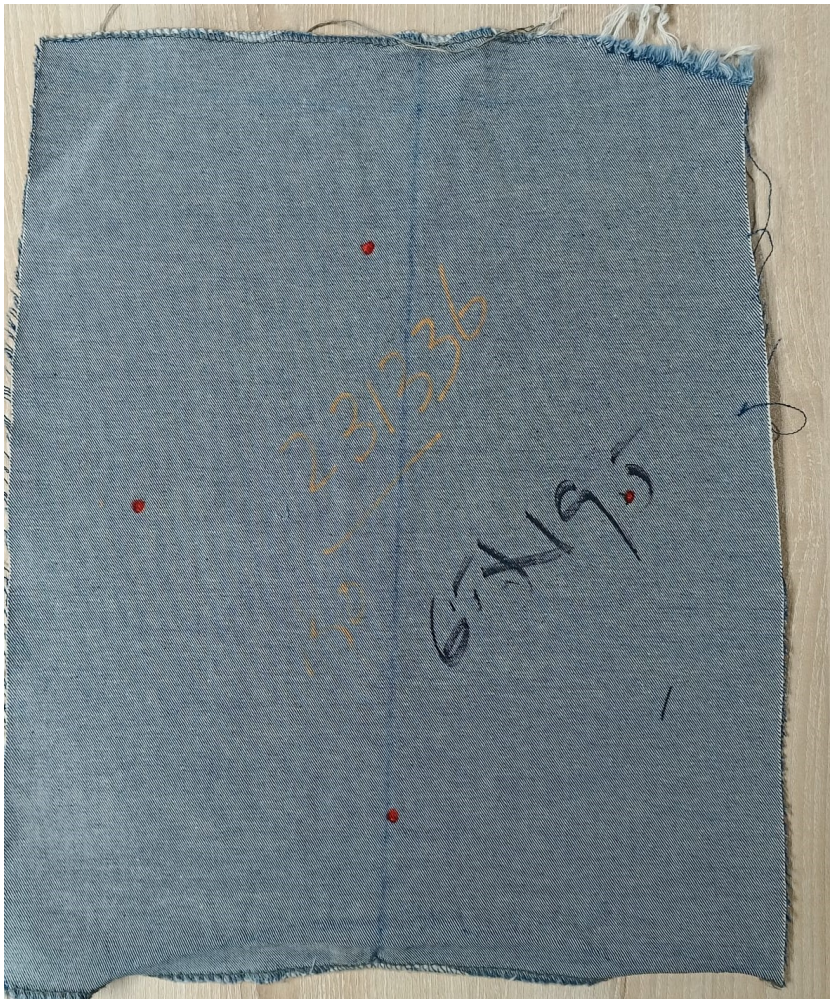

Figure 11: Original Image

Figure 12: Result

Figure 13: Original Image

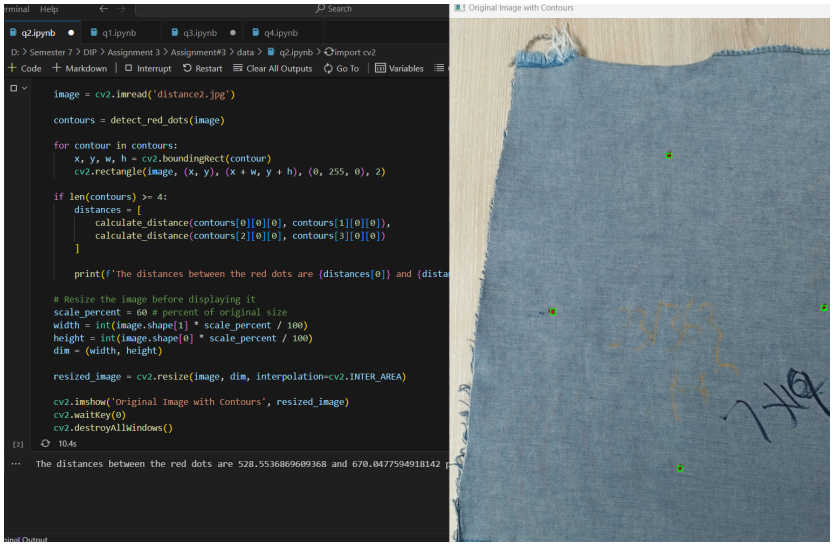

Figure 14: Result

Figure 15: Original Image

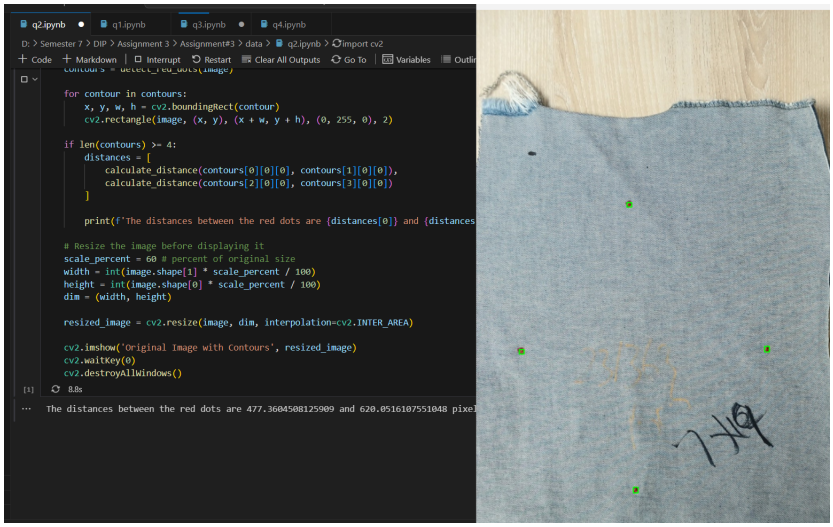

Figure 16: Result

Question 3: Gap Detection

0.5 Steps:

- Import the OpenCV, NumPy, and skimage libraries.
- Read the image using skimage's `io.imread` function.
- Rescale the image intensity using `exposure.rescale_intensity`.
- Convert the image data type to 8-bit unsigned integers (`np.uint8`).
- Convert the image to grayscale using OpenCV's `cvtColor`.
- Apply Gaussian blur to the grayscale image using `cv2.GaussianBlur`.
- Apply closing to the blurred image using `cv2.morphologyEx`.
- Perform edge detection using the Canny edge detector (`cv2.Canny`).
- Find contours in the edge-detected image using `cv2.findContours`.
- Filter out small contours to ignore noise.
- Draw the contours on the original image in red using `cv2.drawContours`.
- Visualize the results by displaying the image with drawn contours.
- Wait for a key press (`waitKey`) and close the display window (`destroyAllWindows`).

0.6 Results:

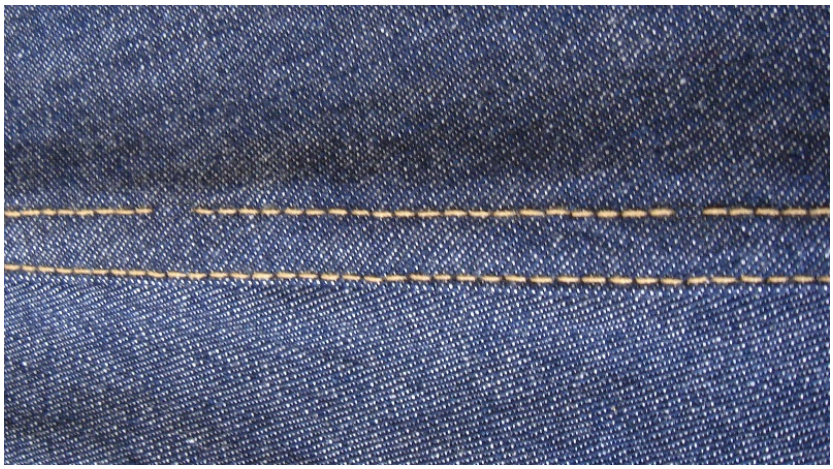

Figure 17: Gap Detection Input

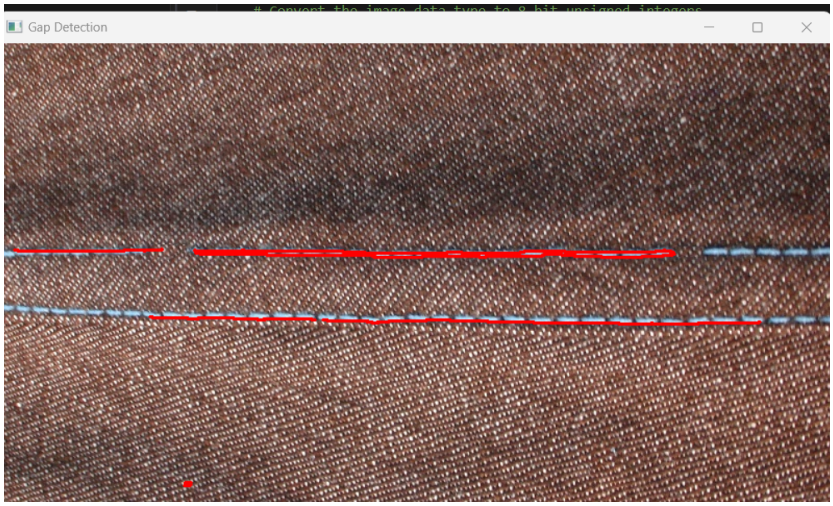

Figure 18: Gap Detection Result

Question 4: Threshold Analysis

0.7 Steps:

- Import the OpenCV library.
- Read a video file (`video_width.mp4`) and store its frames in a list.
- Define a function (`convert_to_bw`) to convert a frame to black and white using Otsu's thresholding.
- Apply the function to all frames to create a list of black and white frames (`frames_bw`).
- Define a function (`count_black_pixels`) to count black pixels in a specific row (e.g., row 200).
- Count black pixels in each black and white frame to create a list (`frames_black_counts`).
- Create a set of unique black pixel counts (`black_counts_set`).
- Create a dictionary (`black_counts_dict`) to store the frequency of each black pixel count.
- Print the dictionary in sorted order of descending black pixel counts.
- Convert the original video to a black and white version using a specified threshold range.
- Write the frame numbers with black pixel counts out of the specified range to a text file (`out_of_range_frames.txt`).
- Print a message indicating where to check for frames with black pixel counts out of the specified range.

0.8 Results:

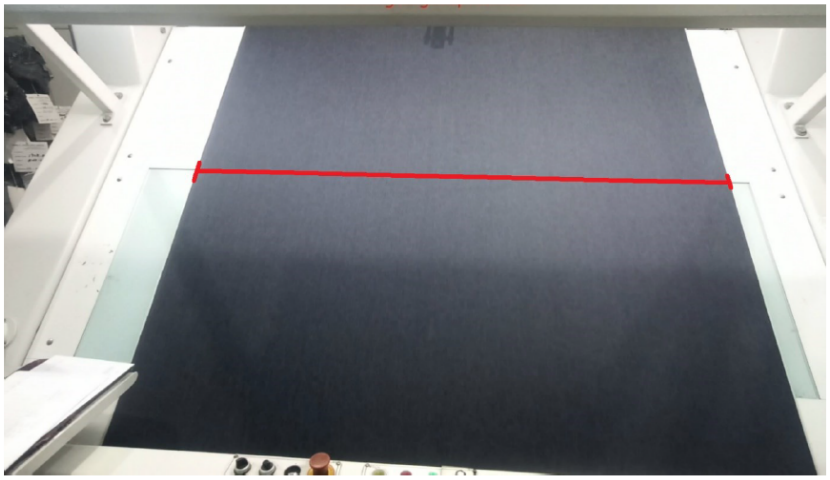

Figure 19: Example Black and White Frame

```
Frame 3 has 1332 black pixels
Frame 4 has 1296 black pixels
Frame 542 has 1277 black pixels
Frame 549 has 1277 black pixels
Frame 553 has 1277 black pixels
Frame 555 has 1277 black pixels
Frame 630 has 1277 black pixels
Frame 631 has 1277 black pixels
Frame 1738 has 1268 black pixels
Frame 1740 has 1268 black pixels
Frame 1744 has 1268 black pixels
Frame 1745 has 1268 black pixels
Frame 1751 has 1268 black pixels
Frame 1757 has 1268 black pixels
Frame 1761 has 1268 black pixels
Frame 1762 has 1267 black pixels
Frame 1763 has 1267 black pixels
Frame 1764 has 1268 black pixels
Frame 1765 has 1268 black pixels
Frame 1766 has 1268 black pixels
Frame 1767 has 1268 black pixels
Frame 1768 has 1268 black pixels
Frame 1769 has 1268 black pixels
Frame 1773 has 1268 black pixels
Frame 1774 has 1268 black pixels
Frame 1775 has 1268 black pixels
Frame 1776 has 1268 black pixels
Frame 1790 has 1268 black pixels
Frame 1791 has 1268 black pixels
Frame 1807 has 1268 black pixels
Frame 1809 has 1268 black pixels
Frame 1810 has 1268 black pixels
Frame 1811 has 1267 black pixels
Frame 1812 has 1267 black pixels
Frame 1817 has 1267 black pixels
Frame 1818 has 1268 black pixels
Frame 1880 has 1268 black pixels
```