UI Component Comparison - Technical Report

Overview

This project presents a visual comparison system for detecting differences between two UI screenshots. The pipeline detects UI components and highlights missing, misplaced, or overlapped elements.

Chosen Pipeline

- Preprocessing using edge detection (Canny) and morphological operations (dilate + close)
- Component detection using contours
- Comparison based on Euclidean distances between bounding box centroids
- Overlap detection using Intersection over Union (IoU)
- Annotation using OpenCV

Why This Pipeline?

This rule-based approach is:

- **Lightweight** and requires no model training
- **Transparent**, making it easy to interpret component differences
- **Accurate** for layout differences even with slight visual changes
- **Flexible** to component-based analysis and modifications

Comparison with Alternatives

1. Pixel Matching
- Compares every pixel of both images.
- **Drawback**: Too sensitive - even small rendering differences or minor shifts trigger false mismatches.

- **Our Pipeline**: Ignores pixel noise and focuses on structural layout differences.

2. SSIM (Structural Similarity Index)
- Computes visual similarity using luminance, contrast, and structure.
- **Drawback**: Gives a single similarity score; does not localize differences.
- **Our Pipeline**: Gives both component-wise matching and visual score with bounding boxes.

3. Template Matching (OpenCV)
- Matches regions using a sliding window template.
- **Drawback**: Fails with resolution/scale differences and layout shifts.
- **Our Pipeline**: Robust to slight misalignments and does not assume fixed templates.

4. ORB + Homography + SSIM
- Uses feature descriptors and geometric transformation.
- **Drawback**: Complex, computationally heavy, and unreliable for flat UI elements with fewer features.
- **Our Pipeline**: Simpler and better suited for structured UI with predictable layouts.

5. Deep Learning (e.g., YOLO for UI Components)
- Uses a trained object detection model for UI elements.
- **Drawback**: Requires large annotated dataset, heavy training cost, and not portable to unseen UIs.
- **Our Pipeline**: No training needed, fully unsupervised, and generalizable.

Conclusion

The chosen component-based pipeline offers a balance of performance, accuracy, and simplicity. It is ideal for automated UI regression testing, layout validation, and anomaly detection without the overhead of deep learning or template matching.