Imaduddin Sheikh

Room Plan Component Detection Using Template Matching and Image Processing: A Comprehensive Report

1. Introduction

This project involves identifying and counting specific components within a room plan using template matching and image processing techniques. The room plan contains symbols representing various components (e.g., lighting fixtures, emergency signs), and the goal is to accurately detect these symbols and report the number of matches for each component.

The project began with an initial idea of parsing component symbols directly from a rasterized PDF using OCR (Optical Character Recognition) and progressed to an image-based template matching method due to the challenges faced with OCR.

2. Problem Definition

The primary task of this project is to:

1. Load a room plan image.
2. Identify and match predefined components (provided as cropped icons) within the room plan.
3. Count and report the occurrences of each component.
4. Ensure that symbols with slight variations (but representing the same component) are grouped and counted as the same.

This required developing a solution that can handle image processing, template matching, and non-maximum suppression to eliminate duplicate detections.

3. Initial Plan and Challenges

Initially, I planned to use a combination of the `pdf2image` library and Tesseract OCR to extract the components directly from a rasterized PDF. This approach involved:

- **Rasterizing the PDF**: Using the `pdf2image` library to convert the PDF containing the room plan into images.
- **Applying OCR**: Using Tesseract OCR to read and extract symbols and component names from the converted images.

However, several issues made this approach infeasible:

- **OCR Limitations**: Tesseract OCR struggled to accurately read the symbols and tables from the rasterized PDF. The symbols were often complex shapes rather than text, and Tesseract was unable to parse them correctly.
- **Table Parsing**: Tesseract had difficulty interpreting the tabular format of component names and associated symbols, leading to errors in data extraction.
- **Inconsistent Results**: The extracted text was often garbled or inaccurate, making it impossible to rely on this method for component detection.

Given these limitations, I pivoted to a more image-focused approach using template matching.

4. Revised Approach: Template Matching

The revised approach involved manually cropping the symbols (representing components) from the room plan and using OpenCV's template matching to detect them in the room plan image. The steps for this approach were:

1. **Manual Cropping**: I manually cropped the symbols from the room plan and saved them as individual images (templates).
2. **Template Matching**: Using OpenCV's `cv2.matchTemplate()` function, I matched these templates against the room plan image to detect occurrences of each component.
3. **Non-Maximum Suppression**: To handle multiple detections of the same component in proximity, I implemented a non-maximum suppression (NMS) function to filter out redundant matches.
4. **Mapping Components with Variations**: Some components had multiple symbols with slight variations (e.g., different orientations or sizes). I mapped these variations to the same component category using a dictionary, ensuring they were counted as a single entity.

5. Methodology

5.1 Loading Room Plan and Components

The project starts by loading the room plan image using OpenCV's `cv2.imread()` function. If the room plan image is not successfully loaded, the program logs an error and exits.

5.2 Template Matching

For each component (represented by a cropped symbol):

- **Sliding Window with Image Pyramid**: A sliding window approach was used, combined with an image pyramid to handle scaling variations in the room plan. This allows the program to match templates at different scales, as components might appear smaller or larger depending on their placement within the room plan.
- **Thresholding**: A correlation threshold (set at 0.8) was used to determine if a match is considered valid. Matches above the threshold were considered potential detections.

5.3 Non-Maximum Suppression

Non-maximum suppression was applied to reduce overlapping detections. This ensures that only one detection is counted for each component instance, even if multiple matches occur within proximity due to the sliding window.

5.4 Component Mapping

The component symbols with variations were mapped to the same component name. For example:

- "Emergency Light - Wet Rated" had two slightly different symbols, but both were mapped to the same component category.
- Multiple variations of recessed lighting were also grouped together.

Here is an example of the component mapping:

```
components = {
    "building_entry_sconce": "Building Entry Sconce",
    "ceiling_fan_integrated_light_kit": "Ceiling Fan w/ Integral Light Kit",
    "damp_rated_exit_sign_universal_double_sided": "Damp-Rated Exit Sign -
Universal Double Sided",
    "damp_rated_exit_sign_universal_single_sided": "Damp-Rated Exit Sign -
Universal Single Sided",
    "emergency_light_wet_rated_1": "Emergency Light - Wet Rated",
    "emergency_light_wet_rated_2": "Emergency Light - Wet Rated",
    # More component mappings...
}
```

5.5 Final Count and Report Generation

The match counts for each component were saved to a text file (`match_counts.txt`). This file contains the component name, and the number of occurrences detected in the room plan.

6. Challenges and Solutions

6.1 OCR Issues

As mentioned, Tesseract OCR proved unreliable in parsing the complex symbols and tables from the rasterized PDF. After several attempts to tune the OCR parameters and improve recognition, it became clear that OCR was not the right tool for this task.

Solution: I switched to manually cropping symbols and performing image-based template matching, which provided far more reliable results.

6.2 Handling Symbol Variations

Some components had multiple variations (e.g., different orientations or slight design differences) that needed to be counted as the same component.

Solution: I created a mapping of these variations to a single component name, ensuring that all symbols were grouped correctly.

6.3 Multiple Detections of the Same Component

Due to the nature of template matching, the same component could be detected multiple times, especially when the window size or pyramid scaling led to overlapping matches.

Solution: I implemented non-maximum suppression to filter out redundant matches and ensure each component was only counted once.

7. Code Functionality

The code was structured as follows:

- **Main Program**: Iterates through each icon (component) in the `components` directory and performs template matching against the room plan.
- **Template Matching**: Uses OpenCV's `cv2.matchTemplate()` to locate symbols in the room plan image.
- **Non-Maximum Suppression**: Filters out redundant matches based on proximity.
- **Output**: The final match counts are written in a text file.

```
for icon_file in os.listdir(icons_dir):
    result = process_icon(icon_file)
    if result:
        component_name, count = result
        if count > 0:
            match_counts[component_name] += count
```

The program runs efficiently, processing each component sequentially and logging the results. The final output report contains the component name and the number of times it was found in the room plan.

8. Deliverables

1. **Source Code**: All source code files used in the program, including the template matching and non-maximum suppression logic, are provided.
2. **Instructions**: Clear instructions on how to set up and run the program are included in a separate document.
3. **Report**: This report details the methodology, assumptions, challenges, and how the code works.
4. **Output**: The result file `match_counts.txt` contains the count of each component found in the room plan.

9. Conclusion

This project demonstrated the effective use of image processing techniques for template matching in a real-world application. Despite the initial setbacks with OCR and PDF parsing, the final solution provided accurate and reliable results by focusing on image-based detection. The manual cropping of symbols, combined with OpenCV's template matching and non-maximum suppression, allowed for the precise detection of components, even when symbols varied slightly.

This approach could be expanded in future iterations by training a more robust model for symbol detection or automating the cropping process further. However, given the constraints and challenges, the solution implemented here successfully met the project's goals.