Project Documentation

# 1. Introduction

The Carpet and Floor Overlay System is a computer vision and image processing pipeline developed to simulate the placement of carpets and floor textures on room images. This system leverages both traditional image processing and deep learning-based segmentation methods to achieve realistic overlays. The main objective is to allow users to visualize carpets or textures in interior settings before making physical decisions.

# 2. Methodology

The system is composed of a Flask server that exposes multiple endpoints for performing different overlay tasks. Input images are encoded in base64 format and sent to the server, which returns modified images also encoded in base64. The major steps include:  
1. Preprocessing of input room and overlay images.  
2. Floor region segmentation using deep learning (MaskFormer).  
3. Carpet transformation into ellipse or trapezoid shapes for perspective realism.  
4. Texture tiling and perspective warping for computational floor overlay.  
5. Binary masking and final image compositing.

# 3. Code Explanation

## 3.1 app.py

This is the main Flask application that runs the server. It exposes the following API endpoints:  
- /overlayCarpet: Applies a carpet overlay (ellipse or trapezoid).  
- /overlayFloor: Overlays a floor texture using a deep learning model (MaskFormer).  
- /overlayFloorComputational: Overlays floor texture using geometric computation.  
Each endpoint receives base64-encoded input images, performs processing using the corresponding module, and returns the base64-encoded result.

## 3.2 carpet\_circle.py

Transforms carpet images into elliptical versions using geometric warping techniques. This simulates the perspective view of a circular carpet placed on a floor surface.

## 3.3 carpet\_working.py

Handles the trapezoidal warping of carpet images to simulate the carpet being placed on a slanted surface. Uses contours, edge detection, and homography matrix computation for perspective transformation.

## 3.4 convert\_binary.py

Creates binary masks of room and overlay images. These masks help in isolating foreground/background and aid the masking process during overlays.

## 3.5 find\_centroid.py

Finds the centroid of the detected floor region using contour-based moments. This is used to position carpets correctly in the overlay process.

## 3.6 floor\_mask\_model.py

Loads the MaskFormer deep learning model from HuggingFace and uses it to segment floor regions in room images. The segmentation mask is then used for further processing like texture tiling and masking.

## 3.7 floor\_overlay.py

Performs the full computational pipeline for overlaying floor textures using perspective warping. This includes tiling the texture, computing the homography, transforming the texture, and compositing with the original image.

## 3.8 mask\_room\_image.py

Uses the segmentation mask obtained from floor\_mask\_model to isolate the floor area in the room image. Returns a processed mask that can be used for texture placement.

## 3.9 overlay.py

Acts as a high-level wrapper to combine ellipse and trapezoid carpet warping with the original room image. Also supports binary masking and final compositing into a single image.

## 3.10 scale\_and\_overlay.py

Handles the resizing and alignment of carpets to match room dimensions. Scales the carpet based on room features and places it at the computed centroid location.

## 3.11 test\_app.py

A batch testing client script. Sends base64-encoded images to the API endpoints, processes responses, and saves output images. Supports testing across all combinations of rooms, carpets, and designs using `itertools.product`. Contains utility functions to handle encoding, decoding, and communication with the server.

**4. API Usage with Postman**

Follow these steps to test API endpoints using Postman:

1. Start the Flask server: python app.py

2. In Postman, create a new POST request.

3. Use endpoint http://127.0.0.1:5000/<endpoint>, where <endpoint> is one of: /overlayCarpet, /overlayFloor, or /overlayFloorComputational.

4. Set headers to Content-Type: application/json.

5. Add the following JSON in the body (raw):

Example JSON for /overlayCarpet:

{  
 "room\_image": "BASE64\_ROOM\_IMAGE\_STRING",  
 "carpet\_image": "BASE64\_CARPET\_IMAGE\_STRING",  
 "overlay\_type": "ellipse"  
}

Example JSON for /overlayFloor:

{  
 "room\_image": "BASE64\_ROOM\_IMAGE\_STRING",  
 "design\_image": "BASE64\_DESIGN\_IMAGE\_STRING"  
}

Example JSON for /overlayFloorComputational:

{  
 "room\_image": "BASE64\_ROOM\_IMAGE\_STRING",  
 "design\_image": "BASE64\_DESIGN\_IMAGE\_STRING",  
 "height\_mul": 2,  
 "width\_mul": 3  
}

# 5. Output Management

The system stores generated images in the following locations:  
- API calls save to `final\_out/`  
- Batch processing (`test\_app.py`) saves to `batch\_outputs/`  
The results can be visually inspected for performance and quality verification.