Introduction

Image rectification plays a crucial role in correcting distortions caused by affine transformations due to viewpoint changes. This process finds widespread application in various domains, including document scanning and digital photography of artwork. The primary objective of this task is to develop an image rectification algorithm capable of transforming input images containing objects with rectangular boundaries against a monochromatic background. The ultimate aim is to produce rectified images where the boundary of the object of interest aligns either horizontally or vertically within the image frame. This report details the implementation of such an algorithm, highlighting the steps involved and the strategies employed to achieve accurate image rectification.

The approach for rectifying the input image involves a systematic multi-step process:

1. **Obtaining the Edgemap**: Using the Canny edge detector, the input image is processed to highlight the edges and contours of the object of interest. This step enhances the visibility of the object's boundaries, facilitating accurate corner detection.
2. **Contour Detection and Corner Extraction**: Contour detection is performed on the edgemap to identify the outlines of the object. From these contours, approximate corner points are extracted to delineate the shape and boundaries of the object accurately.
3. **Perspective Transformation and Backward Warping**: A perspective transform is applied to the image, leveraging the extracted corner points to rectify any distortions caused by viewpoint or affine transformations. Backward warping is employed to ensure that each pixel in the rectified image corresponds to its correct position in the original image, preserving spatial relationships and minimizing distortion.

Figure 1: The column left shows the input image and right column shows the output images after the full image rectification process

Problem-solving Methodology

Obtaining the Edgemap

1. To begin the image rectification process, the input image is first converted to grayscale to simplify subsequent operations. Additionally, a Gaussian blur is applied to the grayscale image to smoothen it and reduce noise. The application of Gaussian blur is essential as it helps to suppress high-frequency noise and minor variations in pixel intensity, resulting in a cleaner image.
2. Subsequently, the opening operation (a morphological operation) is performed on the blurred image to further eliminate noise and fine details. The opening operation, which is a combination of dilation followed by erosion, effectively removes small-scale artifacts while preserving the overall structure of larger objects. This step is crucial in preparing the image for edge detection.
3. Finally, edges are detected using the Canny edge detector algorithm. Canny edge detection is a multi-stage process that involves gradient computation, non-maximum suppression, and edge tracking by hysteresis. By detecting edges, we identify significant transitions in pixel intensity, which often correspond to object boundaries or significant features in the image.

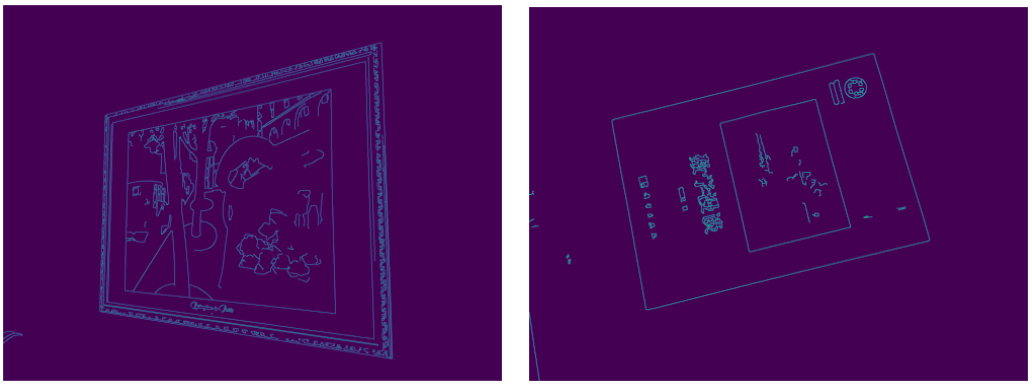

Figure 2: These are the edgemaps produced after removing noise.

Contour Detection and Corner Extraction

1. After obtaining the edge map, contours are extracted using the findContours function provided by the OpenCV library. Contours represent the boundaries of connected components in the edge map, potentially outlining the object of interest. Among the detected contours, the largest contour is selected as it likely corresponds to the boundary of the prominent object in the image.
2. To approximate the contour to a polygon, the approxPolyDP function is applied. This function reduces the number of vertices of the polygon while preserving its essential shape. By specifying an appropriate epsilon value, which controls the approximation accuracy, a simplified polygon closely resembling the original contour is obtained.
3. Finally, the corner points of the polygon are extracted, providing crucial landmarks for subsequent image rectification.

Figure 3: These images show the largest contours detected and the corners approximated from the process. The contours are shown in green lines and the corners are plotted as red dots.

Perspective Transformation and Backward Warping

1. The process of obtaining the warped image involves several key steps. Initially, the dimensions of the object are determined based on the extracted corner points. By measuring the height and width of the object boundary, we establish its spatial extent within the image. This step ensures that the rectified image will accurately represent the size and shape of the object.
2. Next, target dimensions are defined to specify the desired output size of the rectified image. Typically, the output dimensions are chosen to align with the dimensions of the input image, ensuring consistency in the output format. Additionally, it is a good practice to align the center of the rectified image with the center of the rectangular frame after warping. This approach helps to avoid cutting off parts of the object of interest and ensures that the rectified image maintains its overall balance and symmetry.
3. With the dimensions of the object and the target defined, a perspective transform matrix is estimated. This matrix encapsulates the geometric transformation required to rectify the image, effectively mapping the irregular object boundary to a rectangular shape. The perspective transform matrix is calculated based on the relationship between the corner points of the object and their corresponding positions in the target output.
4. Finally, the perspective transform is applied to the input image using the backward warping method. In backward warping, each pixel in the output image is mapped back to its corresponding position in the input image using the inverse transformation defined by the perspective transform matrix. This approach ensures that each pixel in the output image is derived from a specific location in the input image, preserving the spatial relationship between pixels and preventing gaps or overlaps in the rectified image. The resulting warped image accurately represents the object of interest, with its boundary aligned either horizontally or vertically within the frame, as desired.

```python
# Define object height and width
top_r,top_l,bot_l,bot_r = corners # same format across different images
# because of the perspective of different images, the edge nearer to camera will appear bigger
# as such, should select the smaller of the 2 parallel edges
object_height = min(bot_l[1]-top_l[1],bot_r[1]-top_r[1])
object_width = min(bot_r[0]-bot_l[0],top_r[0]-top_l[0])

# Define the output rectangular points (desired output)
output_height = img.shape[0]
output_width = img.shape[1]
img_centre_x,img_centre_y = output_width//2,output_height//2
# assign the coordinates of the target corners accordingly
target_top_r = [img_centre_x+object_width//2,img_centre_y-object_height//2]
target_top_l = [img_centre_x-object_width//2,img_centre_y-object_height//2]
target_bot_r = [img_centre_x+object_width//2,img_centre_y+object_height//2]
target_bot_l = [img_centre_x-object_width//2,img_centre_y+object_height//2]
target = np.array([target_top_r,target_top_l,target_bot_l,target_bot_r], dtype=np.float32)

# Estimate the perspective transform matrix
perspective_matrix = cv2.getPerspectiveTransform(corners.astype(np.float32), target)

# Warp the image using the perspective transform
warped_image = cv2.warpPerspective(img, perspective_matrix, (img.shape[1], img.shape[0]), borderMode=cv2.BORDER_REPLICATE)
```

Figure 4: This is a code snippet to show the important logic on ensuring the centre of the object in the new target image coincides with the centre of the entire new target image. This ensures that the rest of the input image is not cutoff after warping. The output image after warping can be seen in Figure 1.

Implementation Tricks

Applying Gaussian noise serves as a crucial preprocessing step in the image rectification process, aimed at enhancing the quality and accuracy of the resulting output. By introducing controlled levels of random noise to the image, Gaussian blur smoothens the pixel intensity variations, effectively reducing the impact of high-frequency noise and minor fluctuations in pixel values.

When Gaussian noise is not applied, the input image may contain artifacts, such as speckles or pixel-level irregularities, which can interfere with edge detection and contour extraction algorithms. These artifacts may lead to inaccurate or incomplete identification of object boundaries, resulting in suboptimal corner detection and perspective transformation.

Conversely, by applying Gaussian noise prior to edge detection, the input image undergoes a smoothing effect that helps to mitigate the influence of noise while preserving important image features. The resulting edgemap exhibits clearer and more defined edges, facilitating more accurate contour detection and corner extraction.

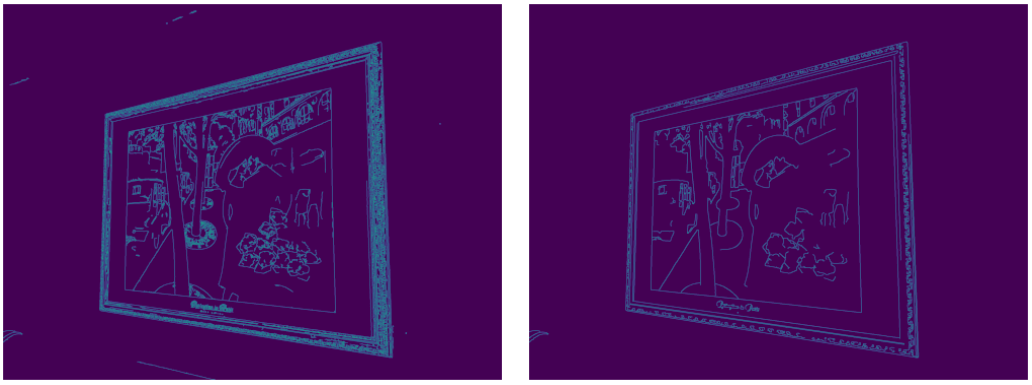

Figure 5: Left image: No Gaussian Blur, Right Image: Gaussian Blur applied
We can see that many more edges corresponding to smaller and more minor fluctuations in pixel values are detected in the left image where the Gaussian blur was not applied.

Figure 6: Left image: No Gaussian Blur, Right Image: Gaussian Blur applied
We can see that the contour on the left is completely wrong as the contour detector fails to detect the correct outer edge due to the noise.

In summary, the implementation trick of applying Gaussian noise before edge detection improves the robustness and reliability of the image rectification process. It enhances the quality of the resulting output by reducing noise-induced artifacts and ensuring more accurate identification of object boundaries, ultimately leading to more precise perspective transformation and rectification of the input image.

Possible Modifications

1. Adaptive Gaussian Blur: Instead of applying a fixed Gaussian blur kernel size, consider implementing an adaptive approach where the blur kernel size is dynamically adjusted based on the local image characteristics. This can help optimize the blur effect for different regions of the image, improving the overall noise reduction and edge preservation.
2. Edge Enhancement Techniques: Experiment with various edge enhancement techniques, such as histogram equalization or gradient-based methods, to further accentuate the object boundaries in the edgemap. Enhancing the edges can lead to more precise contour detection and corner extraction, ultimately improving the accuracy of the rectification process.
3. Contour Refinement Algorithms: Explore advanced contour refinement algorithms to refine the detected contours and improve the accuracy of corner extraction. Techniques such as contour smoothing or curve fitting can help eliminate small irregularities and noise in the contours, resulting in more reliable corner points for perspective transformation.
4. Post-processing Filters: Implement post-processing filters, such as morphological operations or median filtering, to further refine the rectified image and remove any residual noise or artifacts introduced during the transformation process. These filters can help enhance the visual quality of the output image and ensure a smoother transition between pixels.