Journal of Computers Vol. 35 No. 6, December 2024, pp. 97-107
doi: 10.53106/199115992024123506008

A Fusion Algorithm Based on Deep Learning for Panoramic Image

Jianwei Chen[1], Quan Du[2], and Ling-Ju Hung[3*]

[1] Yangtze University College of Arts and Science, China
279497405@qq.com

[2] School of Computer Science, Yangtze University, China
202004010@yangtzeu.edu.cn

[3] Department of Creative Technologies and Product Design, National Taipei University of Business, Taiwan
ljhung@ntub.edu.tw

Received 16 October 2024; Revised 1 November 2024; Accepted 7 November 2024

Abstract. Traditional image fusion algorithms often struggle with slow processing speeds and suboptimal results, particularly when handling non-planar images. In this paper, we present a novel deep learning-based approach for panoramic image fusion. We begin by detailing our dataset construction and preprocessing techniques. To enhance the model's capability with non-planar images, we apply the Thin Plate Spline (TPS) deformation algorithm, allowing effective panoramic fusion across complex image structures. The model architecture is based on a convolutional neural network (CNN) framework, integrated with up- and down-sampling modules to accurately and efficiently capture image features, resulting in higher-quality fusion outcomes. Experimental results demonstrate that this deep learning approach achieves faster fusion speeds and higher quality compared to traditional methods.

Keywords: image mosaic, deep learning, panoramic picture, convolutional neural network

1 Introduction

In daily life, camera limitations often prevent capturing an entire scene due to restricted focal length [1]. While adjusting the focal length or reducing the frame size can address this, such changes often compromise image resolution. Panoramic image fusion technology offers a solution by merging multiple overlapping images into a single, high-resolution panoramic view, providing a broader perspective without losing clarity. This approach is especially beneficial for low-pixel cameras, making high-quality, wide-angle images more accessible and cost-effective for everyday users. However, achieving the desired image resolution remains challenging, particularly due to equipment and budget constraints. Finding ways to enhance image resolution affordably could significantly benefit fields like medicine, biology, and artificial intelligence, where high-quality imaging is crucial.

Image fusion technology has garnered substantial interest globally, with ongoing research aiming to improve efficiency and conserve computational resources. Broadly, image fusion techniques are divided into traditional and deep learning-driven approaches. Traditional methods include feature-based [2], grayscale-based, model-based [3], and transform domain-based methods [4-6]. Each approach has distinct strengths, and researchers are increasingly exploring deep learning to achieve faster, higher-quality fusion results with greater adaptability.

With recent advancements in deep learning, image fusion techniques based on deep learning have seen notable progress and refinement. However, these methods require extensive datasets for training, which can be difficult to gather in some applications. Furthermore, fusion results are highly dependent on the training data quality, making it essential to develop a more efficient, adaptable, and versatile deep learning-driven fusion technique. This paper is built upon addressing these challenges.

In this paper, we apply deep learning to panoramic image fusion, demonstrating through experiments that this approach surpasses traditional fusion methods in terms of feature extraction, computational efficiency, and fusion quality. We address three primary objectives. First, it investigates the implementation and performance of the traditional SIFT algorithm in image fusion. Second, it introduces a deep learning-based approach for image fusion,

* Corresponding Author

presenting both its implementation and optimization methods. This deep learning method offers distinct advantages over traditional techniques: it automatically extracts image features, leading to more precise alignment and fusion; it can handle complex scenes and dynamic elements to create seamless transitions; and it includes a visualization component, providing an interactive interface for user-friendly application.

2 Related Works

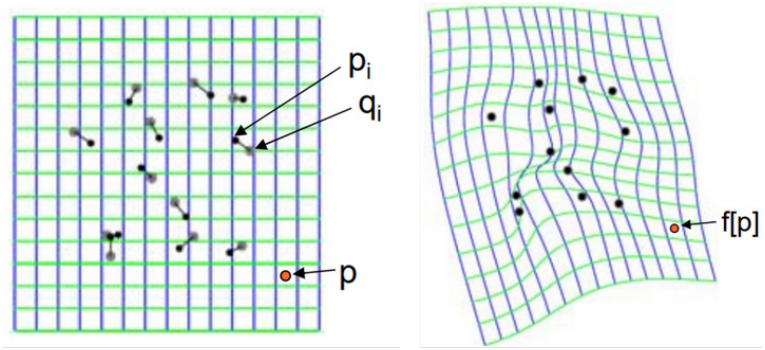
Feature extraction is essential in image fusion, as it directly impacts the quality of the final fused image. The main goal is to identify and match shared features across multiple images, allowing for precise alignment and a cohesive fusion result. In the following subsections, we review key methods integral to our approach: the SIFT algorithm, TPS transformation, neural networks, and convolutional neural networks.

2.1 SIFT Algorithm

The Scale-Invariant Feature Transform (SIFT) algorithm, introduced by David G. Lowe in 1999 and refined in 2004 [7, 8], is a widely used feature extraction and characterization method in computer vision. SIFT identifies "keypoints" within an image and generates descriptors that remain invariant to changes in scale, rotation, and lighting. This robustness makes SIFT particularly valuable for tasks such as image matching and object recognition, where accurate alignment and feature consistency are essential.

2.2 TPS Transformation

The Thin Plate Spline (TPS) function [9] is a widely used interpolation method in image alignment, particularly effective for image matching tasks due to its two-dimensional interpolation approach. In the alignment process, a set of matching points is first identified across two images. The TPS algorithm is then applied to map these points to their corresponding positions in the other image, enabling precise and smooth alignment.



(a) The control point p before transformation; (b) The corresponding point $f[p]$ after transformation and the resulting deformation of the entire plane due to the transformation, showing positional changes of surrounding points.

Fig. 1. The basic process of the TPS interpolation technique

Fig. 1 illustrates the fundamental process of the TPS interpolation technique. In Fig. 1(a), point p represents the control point's position before transformation, while point $f[p]$ in Fig. 1(b) shows its position after transformation. This transformation deforms the entire plane around the control point. As shown in Fig. 1(b), the TPS technique aims to calculate the positional change of each point on the surface resulting from this transformation.

2.3 Neural Network

A neural network is an artificial computational model designed to emulate the connections between biological neurons. It is widely used in machine learning and artificial intelligence. By processing large amounts of input data, neural networks can extract complex features and use these to perform specific tasks, generating the desired output. The basic architecture of a neural network is illustrated in Fig. 2.

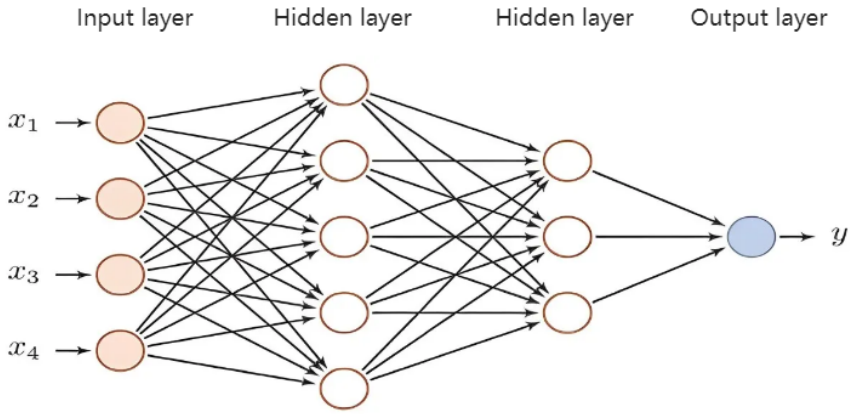


Fig. 2. Basic architecture of a neural network

2.4 Convolutional Neural Network

Convolutional neural networks (CNNs) [10] differ from fully connected networks by employing localized connections through convolutional and pooling layers, which reduces redundancy and decreases computational demands. In CNNs, neurons connect only to neighboring regions, enabling efficient processing of high-dimensional data. A standard CNN structure includes an input layer, an output layer, and multiple hidden layers with alternating convolutional and pooling functions. This layered configuration allows CNNs to effectively capture and learn features from images, making them highly suitable for tasks like classification, recognition, and the processing of complex data.

3 Methodology

In this section, we present the techniques employed to achieve panoramic image fusion using a deep learning approach. Our methodology is organized into three primary stages: first, dataset construction, where we outline the selection criteria and organization of images essential for effective model training and testing. We then move to data preprocessing, detailing the steps taken to optimize the dataset for model training, including resizing, normalization, and data augmentation to improve consistency and model generalization. Finally, we introduce our Deep Learning Model-Based Panoramic Image Fusion Algorithm, covering the network architecture and training strategies designed to precisely align and fuse images with overlapping regions. These combined steps provide a structured approach to creating high-quality panoramic image fusion.

3.1 Dataset Construction

As this study employs deep learning, which relies on extensive data, we use a publicly available dataset from the web. The dataset is split into two parts: a training set and a test set. The training set consists of over 20,000 images organized into two folders, while the test set includes around 2,000 images, also organized into two folders. All images are RGB with overlapping regions, allowing for effective feature learning and fusion. Sample images from the dataset are shown in Fig. 3 and Fig. 4.

Fig. 3. Outdoor data

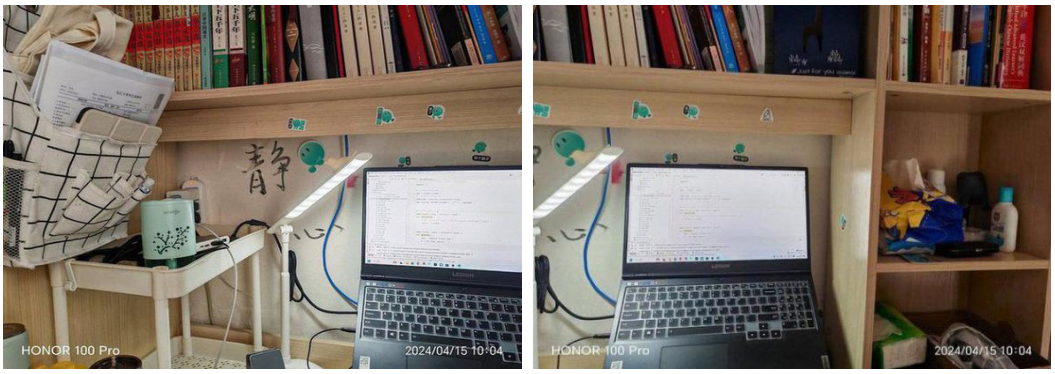


Fig. 4. Indoor data

3.2 Data Preprocessing

Once the dataset is constructed, data preprocessing is essential to enhance the quality of the subsequent deep learning process. Testing has shown that image size and channel count significantly influence deep learning efficiency, so proper preparation of the images is crucial before fusion.

Since deep learning requires intensive training, using large images may slow down the training process, and insufficient computational power can prevent the program from running. Conversely, images that are too small may result in reduced clarity in the final output. To address this, we employ the resize function from the PIL library to resize and compress images, specifying (width, height) along with the Image.ANTIALIAS parameter. Setting Image.ANTIALIAS helps retain image quality during compression.

During preprocessing, the PIL library's size property is used to retrieve the image's height, width, and channel count. RGB images, typically with pixel values ranging from 0–255, are normalized to a 0–1 range for improved processing.

After compression, the images maintain high fidelity to the originals, achieving the intended processing quality and enabling efficient deep learning fusion.

3.3 Deep Learning Model Based Panoramic Image Fusion Algorithm

The deep learning-based panoramic image fusion algorithm consists of two primary stages: image deformation and image fusion. In the first stage, the TPS transformation is applied to deform the image. In the second stage, an enhanced convolutional neural network is used to fuse the deformed images, creating a seamless panoramic result.

TPS Image Deformation. Traditional image fusion techniques often rely on feature extraction algorithms such as SIFT, Harris, and SURF, which work well for images with distinct features and simple geometric compositions. For non-planar scenes, traditional fusion can utilize grid-based models; however, this approach does not suit deep learning-based image fusion. To address this, we apply the Thin Plate Spline (TPS) transformation to the images.

TPS, widely used in applications like facial alignment and expression transformation, is a powerful interpolation method that enables image deformation by converting coordinates. This transformation relies on corresponding keypoints, where one set of keypoints denotes positions in the original image, and another set represents positions in the target template. By solving for TPS parameters, we create a mapping function that aligns each pixel in the original image with its corresponding position in the target template.

This process resembles deforming a flexible plate and projecting it onto the target image, thereby transforming the original image to match the desired structure. Two smooth functions are used here for 2D image transformation:

$$f_{x'}(x,y) = a_1 + a_x x + a_y y + \Sigma N_i = 1 w_i U(\|(x_i, y_i) - (x, y)\|)... \quad (1)$$

$$f_{y'}(x,y) = a_1 + a_x x + a_y y + \Sigma N_i = 1 w_i U(\|(x_i, y_i) - (x, y)\|)... \quad (2)$$

In brief, the coefficients a_1, a_x, and a_y establish a plane that aligns with control points (x_i, y_i) nearest to the target coordinates x' or y', while w_i indicates the influence or weight of each control point. The kernel function, $(\|(x_i, y_i) - (x, y)\|)U(\|((x_i, y_i)U - (x, y))\|)$ measures the distance between the target point and each control point, with the thin plate kernel serving as a typical example. This visualization shows how proximity to a control point (kernel center) impacts each point's transformation effect. After the TPS transformation is applied, the image is warped from its original state (Fig. 5) to the target configuration (Fig. 6).

Fig. 5. Original image before applying the TPS transformation

Fig. 6. Target image after applying the TPS transformation, illustrating the deformation achieved through control point alignment

Convolutional Neural Network Image Fusion. The image fusion algorithm is based on a convolutional neural network framework with two primary components: Net1 and Net2, each further divided into two sub-modules. The detailed architecture is illustrated in Fig. 7 to Fig. 10.

```
self.regressNet1_part1 = nn.Sequential(
    nn.Conv2d(2, 64, kernel_size=3, padding=1, bias=False),
    nn.ReLU(inplace=True),
    nn.Conv2d(64, 64, kernel_size=3, padding=1, bias=False),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(2, 2),

    nn.Conv2d(64, 128, kernel_size=3, padding=1, bias=False),
    nn.ReLU(inplace=True),
    nn.Conv2d(128, 128, kernel_size=3, padding=1, bias=False),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(2, 2),

    nn.Conv2d(128, 256, kernel_size=3, padding=1, bias=False),
    nn.ReLU(inplace=True),
    nn.Conv2d(256, 256, kernel_size=3, padding=1, bias=False),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(2, 2)
)
```

Fig. 7. Part1 of Net1

```
self.regressNet1_part2 = nn.Sequential(
    nn.Linear(in_features=4096, out_features=4096, bias=True),
    nn.ReLU(inplace=True),

    nn.Linear(in_features=4096, out_features=1024, bias=True),
    nn.ReLU(inplace=True),

    nn.Linear(in_features=1024, out_features=8, bias=True)
)
```

Fig. 8. Part2 of Net1

```
self.regressNet2_part1 = nn.Sequential(
    nn.Conv2d(2, 64, kernel_size=3, padding=1, bias=False),
    nn.ReLU(inplace=True),
    nn.Conv2d(64, 64, kernel_size=3, padding=1, bias=False),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(2, 2),

    nn.Conv2d(64, 128, kernel_size=3, padding=1, bias=False),
    nn.ReLU(inplace=True),
    nn.Conv2d(128, 128, kernel_size=3, padding=1, bias=False),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(2, 2),

    nn.Conv2d(128, 256, kernel_size=3, padding=1, bias=False),
    nn.ReLU(inplace=True),
    nn.Conv2d(256, 256, kernel_size=3, padding=1, bias=False),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(2, 2),

    nn.Conv2d(256, 512, kernel_size=3, padding=1, bias=False),
    nn.ReLU(inplace=True),
    nn.Conv2d(512, 512, kernel_size=3, padding=1, bias=False),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(2, 2)
)
```

Fig. 9. Part1 of Net2

```
self.regressNet2_part2 = nn.Sequential(
    nn.Linear(in_features=8192, out_features=4096, bias=True),
    nn.ReLU(inplace=True),

    nn.Linear(in_features=4096, out_features=2048, bias=True),
    nn.ReLU(inplace=True),

    nn.Linear(in_features=2048, out_features=(grid_w + 1) * (grid_h + 1) * 2, bias=True)

)
```

Fig. 10. Part2 of Net2

The fusion algorithm also defines three core modules: the downsampling module, which gradually reduces the feature map size; the upsampling module, which progressively restores the feature map size; and the main network module, which initializes the downsampling and upsampling modules and connects them through a forward propagation process.

```
class DownBlock(nn.Module):
    def __init__(self, inchannels, outchannels, dilation, pool=True):
        super(DownBlock, self).__init__()
        blk = []
        if pool:
            blk.append(nn.MaxPool2d(kernel_size=2, stride=2))
        blk.append(nn.Conv2d(inchannels, outchannels, kernel_size=3, padding=1, dilation=dilation))
        blk.append(nn.ReLU(inplace=True))
        blk.append(nn.Conv2d(outchannels, outchannels, kernel_size=3, padding=1, dilation=dilation))
        blk.append(nn.ReLU(inplace=True))
        self.layer = nn.Sequential(*blk)

        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(m.weight)
            elif isinstance(m, nn.BatchNorm2d):
                m.weight.data.fill_(1)
                m.bias.data.zero_()

    def forward(self, x):
        return self.layer(x)
```

Fig. 11. Downsampling module

As shown in Fig. 11, this module gradually reduces the spatial resolution of the feature map using convolutional and pooling layers, which increases the number of channels as the resolution decreases.

The upsampling module in Fig. 12 employs nearest neighbor interpolation and 1×1 convolution to increase the spatial resolution of the feature map, while additional convolutional layers further refine feature extraction. Fig. 13 illustrates the main network module, which initializes both the upsampling and downsampling modules.

```python
class UpBlock(nn.Module):
    def __init__(self, inchannels, outchannels, dilation):
        super(UpBlock, self).__init__()
        # self.convt = nn.ConvTranspose2d(inchannels, outchannels, kernel_size=2, stride=2)
        self.halfChanelConv = nn.Sequential(
            nn.Conv2d(inchannels, outchannels, kernel_size=3, padding=1),
            nn.ReLU(inplace=True)
        )
        self.conv = nn.Sequential(
            nn.Conv2d(inchannels, outchannels, kernel_size=3, padding=1, dilation=dilation),
            nn.ReLU(inplace=True),
            nn.Conv2d(outchannels, outchannels, kernel_size=3, padding=1, dilation=dilation),
            nn.ReLU(inplace=True)
        )
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(m.weight)
            elif isinstance(m, nn.BatchNorm2d):
                m.weight.data.fill_(1)
                m.bias.data.zero_()
```

Fig. 12. Upsampling module

```python
class Network(nn.Module):
    def __init__(self, nclasses=1):
        super(Network, self).__init__()

        self.down1 = DownBlock(3, 32, 1, pool=False)
        self.down2 = DownBlock(32, 64, 2)
        self.down3 = DownBlock(64, 128, 3)
        self.down4 = DownBlock(128, 256, 4)
        self.down5 = DownBlock(256, 512, 5)
        self.up1 = UpBlock(512, 256, 4)
        self.up2 = UpBlock(256, 128, 3)
        self.up3 = UpBlock(128, 64, 2)
        self.up4 = UpBlock(64, 32, 1)

        self.out = nn.Sequential(
            nn.Conv2d(32, nclasses, kernel_size=1),
            nn.Sigmoid()
        )
```

Fig. 13. Main network module

Visualization Interface. To enhance user convenience, a visual interface was developed, simplifying the operation process. As shown in Fig. 14, the overall workflow is as follows: the user selects the images for fusion and clicks the “Upload” button, uploading them to the system. Once the upload is successful, the user clicks “Generate Perspective View,” prompting the system to begin calculations. After completion, the system notifies the user, who can then view the intermediate results on the display page.

In the final stage, the user initiates the image fusion by clicking “Generate Result,” prompting the system to process and display the fused image on the interface. Additionally, the interface allows users to compare the fused image with the original images for easy evaluation.

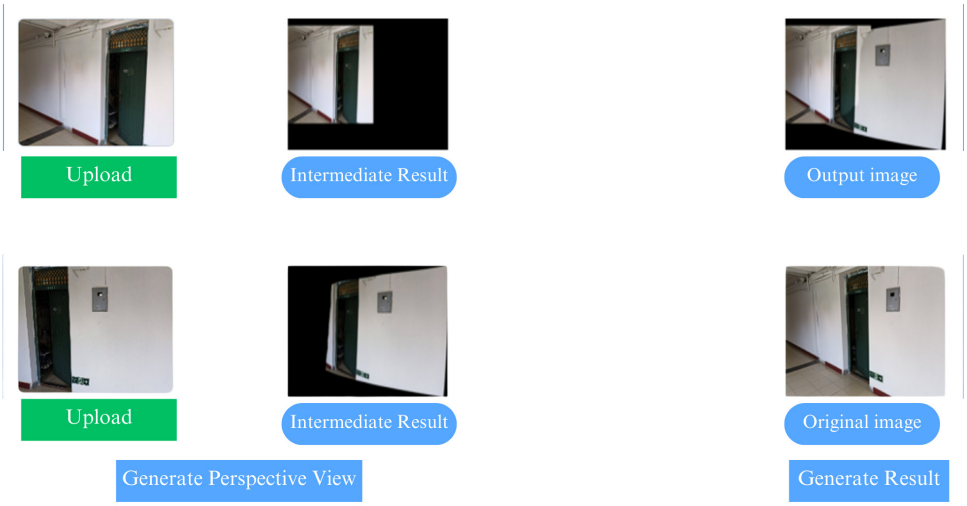


Fig. 14. GUI design

4 Experimental Results

To clearly illustrate the advantages of the proposed method, this section presents comparative experiments using the SIFT algorithm alongside our approach. Here, we analyze and compare the image fusion results achieved by the deep learning-based method and the SIFT algorithm.

4.1 SIFT Algorithm Fusion

Fig. 15 illustrates the feature point matching effect between two images using the SIFT algorithm.

Fig. 15. SIFT fusion effect

4.2 Deep Learning Image Fusion

For the same images, the fusion results using the proposed deep learning method are shown in Fig. 16.

Fig. 16. SIFT fusion effect

4.3 Analysis of Results

The analysis below compares the deep learning-based image fusion method proposed in this paper with the traditional SIFT algorithm for image fusion.

Table 1. Speed comparison of the two methods

Algorithm	Fusion time (sec)
Traditional SIFT algorithm	5.227
Deep learning	1.618

Table 1 shows that the deep learning-based image fusion method is approximately three times faster than the traditional SIFT algorithm. This improvement in processing time is due to the extensive feature computation required by SIFT, whereas the deep fusion approach leverages pre-learned features from the model training phase, significantly reducing time. Additionally, using GPU acceleration with deep learning can further decrease time consumption.

Fig. 15 and Fig. 16 compare the fusion quality of the deep fusion method and the traditional SIFT fusion scheme, with Fig. 15 representing the SIFT approach and Fig. 16 representing the deep fusion approach. It is clear that the image clarity achieved with the deep fusion method surpasses that of SIFT. For instance, specific details such as the door number are distinctly visible in the deep fusion results (Fig. 16) but are unclear in the SIFT results (Fig. 15). This highlights the effectiveness of the deep learning fusion method over traditional techniques.

In summary, the proposed deep learning method significantly outperforms the traditional SIFT approach in both fusion time and quality, offering a promising direction for further research in image fusion algorithms.

5 Conclusion

This paper demonstrates the use of deep learning in panoramic image fusion, structured into two main stages: image deformation and image synthesis. To support these processes, a GUI interface was created to offer users an intuitive view of the image fusion workflow. In the first stage, we used Pytorch to design a deep learning network that deforms input images. To ensure reliable outcomes, we iteratively superimposed and refined features, preserving the optimal iteration as the model for the second stage. The deformation results met necessary quality standards, enabling a smooth transition into the fusion process. In the synthesis stage, we employed a second Pytorch-based network tailored to fuse the processed images with high precision. Due to the substantial data involved, this stage focuses on detailed feature capture, but hardware limitations, including the use of a single GPU, impacted processing speed and quality. Future research could improve results by using multiple GPUs for parallel processing. The GUI interface, while basic, effectively lets users select images for fusion. With further development, it could incorporate real-time camera inputs for live fusion, expanding its practical applications. An application example includes museum security, where panoramic fusion can enhance surveillance coverage, tracking multiple visitors at once and facilitating rapid identification in cases such as theft. This approach could significantly improve artifact protection by enabling more comprehensive monitoring.

References

[1] Y.-S. Chen, Y.-Y. Chuang, Natural Image Stitching with the Global Similarity Prior. in: Proc. B. Leibe, J. Matas, N. Sebe, M. Welling (Eds.), Computer Vision – ECCV 2016. ECCV 2016. Lecture Notes in Computer Science, vol. 9909. Springer.
[2] C. Harris, M. Stephens, A Combined Corner and Edge Detector, in: Proc. the 4th Alvey Vision Conference, 1988.
[3] R. Szeliski, Image alignment and stitching: A tutorial, Foundations and Trends® in Computer Graphics and Vision 2(1) (2007) 1-104.
[4] C.D. Kuglin, E.L. Hines, The Phase Correlation Image Alignment Method, in: Proc. the IEEE Conference on Cybernetics and Society, 1975.
[5] E. De Castro, C. Morandi, Registration of translated and rotated images using finite Fourier transforms, IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-9(5)(1987) 700-703.
[6] B.S. Reddy, B.N. Chatterji, An FFT-Based Technique for Translation, Rotation, and Scale-Invariant Image Registration, IEEE Transactions on Image Processing 5(8)(1996) 1266-1271.
[7] D.G. Lowe, Object recognition from local scale-invariant features, in: Proc. the IEEE International Conference on Computer Vision, 1999.
[8] D.G. Lowe, Distinctive Image Features from Scale-Invariant Keypoints, International Journal of Computer Vision 60(2) (2004) 91-110.
[9] F.L. Bookstein, Principal warps: Thin-plate splines and the decomposition of deformations, IEEE Transactions on pattern analysis and machine intelligence 11(6)(1989) 567-585.
[10] M.D. Zeiler, R. Fergus, Visualizing and understanding convolutional networks, in: Proc. European conference on computer vision, 2014.