ONTARIO TECH UNIVERSITY
FACULTY OF SCIENCE, COMPUTER SCIENCE

Project Group: 31 Dec 8, 2024

Simulation and Modeling Course Project: Simulation of our Local Solar System.

by

Sandeep Virk

Abstract:

This project created a Python-based panoramic stitching tool using SIFT and ORB for feature detection, BF and FLANN for matching, and RANSAC for homography refinement. Images are warped, blended, and stitched seamlessly. A PySimpleGUI interface enables user-friendly interaction and algorithm selection. The tool highlights the practicality of computational photography for creating robust, high-quality panoramas.

Project: `https://github.com/sandeepv6/panoramic-stitching`

1 Introduction

Panoramic image stitching has existed as a computational photography technique which is widely used to create seamless wide-angled images. This is achieved by essentially combining, overlapping, and blending images together. Significantly used in the fields of virtual reality, professional photography, and geographic imaging, it has become a staple tool to use within modern photography. By taking advantage of feature detection, homography estimation, image warping as well as blending, panoramic stitching tools provide visually appealing results.

This project aims to develop a panoramic stitching tool using Python and widely recognized libraries, including OpenCV, NumPy, Matplotlib, and PySimpleGUI. This tool enables users to upload images, then select advanced feature detection algorithms of SIFT or ORB as well as between the image matching of Brute Force or FLANN. Then, using the power of homography, warping, and blending, it achieves the results of a panoramic stitched image. Additionally, the project emphasizes creating a visually appealing interface through PySimpleGUI, allowing for intuitive navigation and real-time feedback, but it also paired with simple command-line functionality to give users a preference.

The development of this tool serves as a learning opportunity to explore key concepts in computer vision and computational photography. The project covers advanced topics such as feature detection, homography estimation, image warping, and blending, which are fundamental to many image processing applications. By the end of this project, the goal was not only to create a working panoramic stitching tool but also to gain a deeper understanding of how computational photography techniques can be applied to solve real-world problems.

2 Technical Implementation and Functionality

The core objective of the tool as stated before is to seamlessly stitch multiple images with computational photography techniques. The technical implementation of this process involves several key stages that collectively enable the stitching. These stages include feature matching to identify corresponding points, homography computation to establish geometric relationships, warping to align images, and blending to ensure smooth transitions. Each stage relies on advanced computer photography and vision algorithms or techniques, which are detailed in the

following subsections.

2.1 Feature Detection and Matching

The cornerstone of successful image stitching lies in the precise identification and matching of distinctive features across multiple images. An image feature for matching is a distinct, identifiable point or pattern in an image, such as a corner, edge, or texture, used to establish correspondences between images. Throughout the course we briefly touched on this topic as it is somewhat complex. Feature Detection has more to do with Computer Vision rather than Computer Photography because its primary focus is on interpreting and understanding visual data in a way that computers can process. In my implementation, I explored two prominent feature detection methodologies: Scale-Invariant Feature Transform (SIFT) and Oriented FAST and Rotated BRIEF (ORB). This section will be a brief overview of the equations used in both these algorithms

SIFT was developed by David Lowe. It represents a sophisticated approach to feature detection that transcends traditional image matching techniques. The algorithm constructs a scale space representation of an image. It then systematically identifies keypoints that remain invariant to scale, rotation, and illumination changes. This process involves multiple intricate steps, one of which is extremely important of generating a Difference of Gaussian (DoG) pyramid. It then detects extreme points across different scales.

In contrast with SIFT, the ORB algorithm offers a computationally efficient alternative. By combining the FAST keypoint detector with the BRIEF descriptor, ORB provides a faster yet remarkably effective feature matching approach. This method becomes particularly valuable when processing time is a critical constraint, such as in real-time imaging applications or when working with limited computational resources.

These approaches were neither completely understood or self-implemented within the scope of this project, yet were used to achieve feature detection. They were implemented using the OpenCV library and helped achieve an automatic version of feature detection for a better product of image stitching.

2.2 Homography Estimation

Homography represents the mathematical heart of image registration, describing the projective transformation between two image planes. It is the main staple which allows images to be related to one another essentially. My implementation focuses on the geometric principles that enable smooth image alignment using advanced matrix transformations. The homography matrix H represents a 3x3 transformation that maps points from one image coordinate system to another. Mathematically, this is expressed through the matrix:

$$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

We need minimum 4 correspondences to solve this matrix since it has 8 degrees of freedom. The problem with solving a matrix as such is that it causes numerical issues. So the way around this is to rewrite the homography relationship as homogeneous equations:

$$x_i' = \frac{h_{11}x_i + h_{12}y_i + h_{13}}{h_{31}x_i + h_{32}y_i + h_{33}}$$

$$y_i' = \frac{h_{21}x_i + h_{22}y_i + h_{23}}{h_{31}x_i + h_{32}y_i + h_{33}}$$

We then need to write these equations as matrix-vector products:

$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x_ix_i' & -y_ix_i' & -x_i \\ 0 & 0 & 0 & x_i & y_i & 1 & -x_iy_i' & -y_iy_i' & -y_i \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ \vdots \\ h_{33} \end{bmatrix} = 0$$

This is completed for however many correspondences we choose and we just add to our initial matrix with all coordinate values. Once the matrix is setup it was simple to solve in this projects case using Singular Value Decomposition (SVD). Solving Ax=0 with SVD was simple, as we take V^t and the last column gives us our 3x3 matrix for H. A lot of research here was used to replicate the method the OpenCV uses in order to compute homography matrices, but it was manipulated in a way to match the $Ax = b$ configuration for simple solving as learned in this class.

2.3 Warping

Image warping transforms the source image's geometric representation to align precisely with the reference image. My implementation employs a sophisticated and basic coordinate mapping strategy that goes beyond simple pixel translation. The homography matrix (H) maps points from the input image to the warped (output) space. To determine which source pixels correspond to each pixel in the output image, the inverse homography (H_{inv}) is computed.

Using the inverse homography, the output pixel coordinates (coords) are mapped back to the source image. This step includes: Applying the transformation (H_{inv} @ coords.T).

Normalizing by the third coordinate (src_{coords} / src_{coords}[2, :]) to return to Cartesian coordinates.

Extracting the x and y coordinates (src_{coords}[:2, :].T).

For each valid source pixel, bilinear interpolation is used to estimate its intensity:

The four surrounding integer coordinates $(src_{x0}, src_{x1}, src_{y0}, src_{y1})$ are computed. Clipping ensures these coordinates stay within the input image bounds. The weights (wa, wb, wc, wd) determine how much each of the four surrounding pixel values contributes to the interpolated value. The weight calculation was from an online source which is referenced, and weights provide a better result for the calculated pixel in the interpolation.

The pixel values from the input image (I_a, I_b, I_c, I_d) are combined using the weights to compute the final pixel value at each location. The interpolation equation elegantly blends pixel values based on their proximity, creating a seamless transformation:

2.4 Blending

The final stage of panoramic image creation involves blending multiple warped images. My implementation utilizes a straightforward yet effective linear blending technique that ensures smooth transitions between image regions. The blending process uses a weighted average approach, typically with equal weights to create

a balanced transition. The fundamental blending equation:

$$result = \alpha * image1 + (1 - \alpha) * image2$$

The above equation allows for gradual, imperceptible transitions between overlapping image regions.

3 User Interface and Aesthetics

The project uses PySimpleGUI for a user-friendly interface, offering an easy and intuitive way to select images, feature detectors, and matchers. A variety between PySimpleGUI and a console-line interface helps users have options on how they wish to use the program. The GUI includes options for selecting algorithms and matchers, a ”Stitch Images” button, and a ”Save Image” functionality. This design balances simplicity and functionality, ensuring accessibility for users with varying technical skills.Some advantages of the GUI however are the choice of which algorithm for feature detection, matcher choice, live feedback, as well as being able to save images easily or selecting images easily. It seems clear that the GUI has many advantages over the simple CLI but both are easy to use.

4 Conclusions

In conclusion, the panoramic image stitching project represented more than just a simple technical implementation. It felt like a huge journey to go through with tons of troubleshooting and figuring out converting mathematical techniques and transferring them to Python. It was a trip through intricate landscape of computation photography techniques as well as a dip into computer vision. By combining mathematical transformation methods and user-design, a tool has been created which showcases and helps users through the process of image-stitching.

The tool was tested on a variety of image sets, demonstrating robustness and flexibility in stitching diverse scenarios. Users could seamlessly stitch images, switch between algorithms, and save the output, all within a purposeful interface. The final product not only met but exceeded the initial project goals, showcasing the practical applications of computational photography techniques.

The purpose of implementing these methods was not so that they were the first of their kind, but more about the learning journey. A lot of methods such as finding

the homography, or warping were purposely manually implemented rather than using external libraries simply due to the purpose of learning as much as possible within the Computational Photography and Computer Vision fields. I navigated complex challenges in feature detection, geometric transformation, and image blending, developing a deep appreciation for the mathematical backbone underlying visual processing.

Key learnings emerged throughout the project:

- The critical importance of robust feature matching in image registration
- The mathematical methodology required in geometric transformations of images
- The delicate balance between computational efficiency and image quality
- The significance of user interface design in image displaying and the process of image stitching.

Ultimately, this project has been a rewarding learning experience, providing a deep understanding of both the theoretical and practical aspects of panoramic image stitching. It reinforced the importance of combining algorithmic rigor with thoughtful design to create tools that are not only powerful but also user-friendly. The insights gained from this project will serve as a foundation for future endeavors in computational photography, computer vision, and software development.

5 References

References

[1] GeeksforGeeks. (2024, June 17). *Image stitching with OpenCV*. Retrieved from `https://www.geeksforgeeks.org/image-stitching-with-opencv/`

[2] Kommineni, V. (2020, November 14). *Image stitching using OpenCV*. Medium. Retrieved from `https://towardsdatascience.com/image-stitching-using-opencv-817779c86a83`

[3] Morgan, J. (2023, March). *Panorama creation in OpenCV*. How to Use

OpenCV. Retrieved from https://www.opencvhelp.org/tutorials/advanced/panorama-creation/

[4] Premsingh, P. (2024, August 20). *Image stitching using OpenCV - A step-by-step tutorial*. Medium. Retrieved from https://medium.com/@paulsonpremsingh7/image-stitching-using-opencv-a-step-by-step-tutorial-9214aa4255ec

[5] Rosebrock, A. (2023, June 8). *Image stitching with OpenCV and Python*. PyImageSearch. Retrieved from https://pyimagesearch.com/2018/12/17/image-stitching-with-opencv-and-python/

[6] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision, 60*(2), 91-110. https://doi.org/10.1023/B:VISI.0000029664.99615.94

[7] Hartley, R., & Zisserman, A. (2003). *Multiple view geometry in computer vision*. Cambridge University Press. https://doi.org/10.1017/CBO9780511811685

[8] OpenCV Documentation. (n.d.). Feature detection and description. Retrieved from https://docs.opencv.org/