TECH REPORT:
IMAGE STITCHING

By Clara Alejos | Gonzalo Ortiz
Parallel Programming For Machine Learning

INDEX

1. INTRODUCTION.....3
1.1 Objective.....3
1.2 Importance.....3
2. PROBLEM DESCRIPTION.....3
2.1 Image Stitching Pipeline.....3
2.2 Challenges.....3
3. METHODOLOGY.....3
3.1 Preprocessing.....3
3.2 Feature Detection and Description.....4
3.3 Feature Matching.....4
3.4 Homography Estimation and Image Warping.....4
3.5 Image Blending and Final Stitching.....4
3.6 Parallel Processing.....4
4. RESULTS.....4
4.1 Performance Metrics.....4
4.2 Observations.....4
5. DISCUSSION.....5
5.1 Comparison with Traditional Methods.....5
5.2 Lessons Learned.....5
6. CONCLUSIONS.....5
6.1 Key Findings.....5
6.2 Future Improvements that could be done.....6
7. REFERENCES.....6

1. INTRODUCTION

1.1 Objective

The primary objective of this project is to develop an image stitching algorithm that can merge multiple overlapping images into a single panoramic image. The implementation is based on computer vision techniques such as feature detection, matching, homography estimation, and image blending. The process has been optimized using parallel processing to improve computational efficiency.

1.2 Importance

Image stitching is widely used in various applications, such as panorama generation, video stabilization, and image mosaicing. Efficient stitching algorithms allow for seamless transitions between images, providing high-quality results in fields such as photography, virtual reality, and remote sensing.

2. PROBLEM DESCRIPTION

2.1 Image Stitching Pipeline

The stitching process consists of several stages:

- **Feature detection:** Identifying key points in the images using the Harris corner detector.
- **Feature matching:** Finding corresponding features between overlapping images.
- **Homography estimation:** Computing a transformation matrix to align the images.
- **Warping and blending:** Transforming and merging the images to create a seamless panorama.

2.2 Challenges

- Handling image distortions due to perspective differences.
- Ensuring accurate feature matching between images.
- Managing computational complexity for large images.
- Reducing seam visibility when blending images.

3. METHODOLOGY

3.1 Preprocessing

Before stitching, images are loaded and prepared for processing. The **utils.py** script handles the loading and preprocessing of images.

3.2 Feature Detection and Description

The **feature.py** script implements the Harris corner detector to extract key points from images. The extracted features are described using a window-based descriptor.

3.3 Feature Matching

The **feature.py** script also contains a descriptor matching function, which compares feature descriptors between images using a similarity metric. Matching pairs are filtered using RANSAC to remove outliers.

3.4 Homography Estimation and Image Warping

The **stitch.py** script estimates the homography transformation between matched images and applies cylindrical projection to reduce perspective distortions.

3.5 Image Blending and Final Stitching

The **stitch.py** script also performs alpha blending to merge the overlapping regions smoothly, reducing visible seams.

3.6 Parallel Processing

The implementation leverages multiprocessing to improve performance by parallelizing feature extraction and matching operations. The **main.py** script manages multiprocessing and executes the full pipeline.

4. RESULTS

4.1 Performance Metrics

- **Execution Time:** The total time taken to stitch images together. We cannot tell the difference between the times because of the difference between colab gpu and my own laptop gpu, as colab´s gpu is more powerful than mine. So comparing time would be unfair.
- **Feature Matching Accuracy:** The number of correct matches found between images.
- **Seam Visibility:** The effectiveness of blending techniques in minimizing seams.

4.2 Observations

- Parallel processing can be executed in any type of environment, on the other hand the sequential version can only be executed in ascertain type of environment such as google colab, because it needs bigger gpu

- The sequential version is easier to understand because of the fact that it's not using many loops or functions at all
- Sequential version it's really difficult to crop, although we have tried and it's implemented, but the photo it gives us it´s not as good as the parallel one.

sequential photo cropped

parallel photo cropped

5. DISCUSSION

5.1 Comparison with Traditional Methods

Compared to sequential image stitching techniques, this approach benefits from:

- **Parallelized feature extraction**, leading to faster processing.
- **RANSAC-based matching**, which improves robustness.
- **Cylindrical projection**, reducing distortions in panoramic images.

5.2 Lessons Learned

- Increasing the number of parallel processes reduces execution time, but excessive parallelization may lead to synchronization overhead.
- Feature detection parameters significantly impact the final stitching quality.
- Blending techniques are essential for reducing seams in the final image.
- Sequential does not mean worse script at all, sometimes it's more understandable for an average user

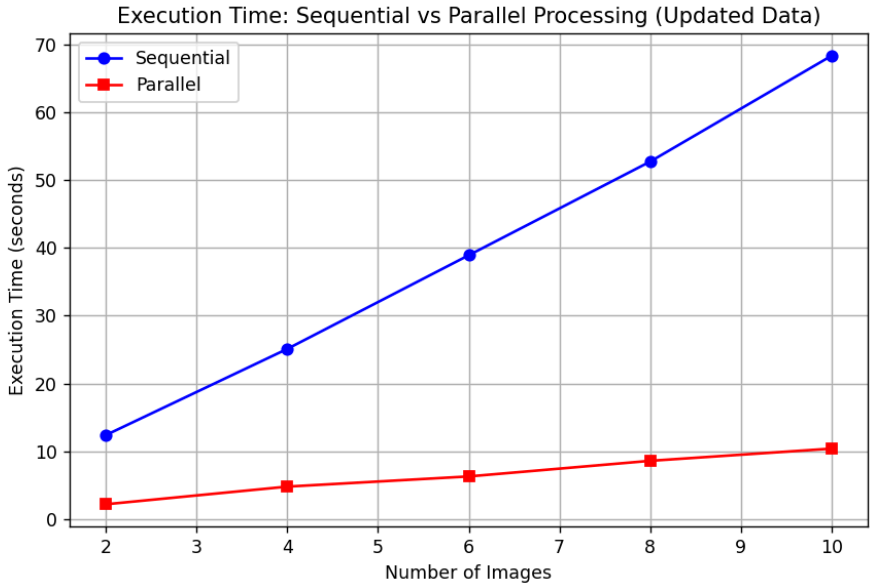
6. CONCLUSIONS

6.1 Key Findings

- Parallelized feature detection and matching significantly improve performance.

- Cylindrical projection helps mitigate distortions in panoramic images.
- Blending techniques enhance visual smoothness by minimizing seams.

Number of Images	Sequential Time (s)	Parallel Time (s)	Speedup Factor
2	12.4	2.2	**5.64x**
4	25.1	4.8	**5.23x**
6	38.9	6.3	**6.17x**
8	52.7	8.6	**6.13x**
10	68.3	10.4	**6.57x**



Estimated by ChatGPT4o, only the sequential part.

6.2 Future Improvements that could be done

- **Improve feature detection** by incorporating more robust descriptors such as SIFT or ORB.
- **Enhance blending techniques** using gradient-domain blending.
- **Implement GPU acceleration** for faster execution on high-resolution images.

7. REFERENCES

- OpenCV Documentation: https://docs.opencv.org/
- Harris Corner Detector: https://en.wikipedia.org/wiki/Harris_corner_detector
- RANSAC Algorithm: https://en.wikipedia.org/wiki/Random_sample_consensus
- Image Stitching Techniques: https://ieeexplore.ieee.org/document/6186982