國立交通大學
資訊工程學系

飛行器應用之 54 智慧 2D 影像拼接技術

Intelligent Image Processing for Aerial 2-D Image Stitching

大　學　生　Andrés Ponce 彭思安

指 導 教 授　Maria Yuang 楊啟瑞

中華民國 109 年 12 月 27 日

1 Abstract

Intelligent Image Processing is one of the subtask of 5G-DIVE Autonomous Drone Scout (ADS) verticals. It aims to intelligently compute drone video stream in the edge to detect persons in need of help, and to provide stitched image of a disaster impacted area. 5G-DIVE project is a collaborative project between the EU and Taiwan to prove the technical merits and business value preposition of 5G technologies in ADS vertical pilot. In this research project, we worked on an improved Aerial 2D-ST solution that leverages the 5G-DIVE platform specifically the IESS to improve 2D-ST. In particular, AI techniques is used as a solution to improve on the existing 2D-ST solution to produce high-quality stitched image but without sacrificing for computation time.

2 Introduction

A panorama in visual art depicts a continuous scene or landscape [1]. The concept of panoramas in photoraphy has existed for centuries. In the middle of the 19^{th} century, landscapes were created by placing daguerrotypes side by side side by side [2]. Panorama images have been popularized in the last years due to their widespread inclusion in smartphones and digital cameras; for example, the iPhone5 introduced panoramic images in 2012 [3]. As processing power has grown, new and increasingly smart solutions have allowed better quality images to be stitched for pleasing results.

In digital photography, a panoramic image refers to a large composite image made of smaller images with overlapping areas. Computer software will look for the optimal way to combine the overlapping areas of the images such that the output panorama exhibits little or no visual artifacts. Image stitching has multiple uses other than recreative or artistic, producing a map of an area from overhead drone image [4], or for medical imaging applications.

3 Problem Description

The problem of focus here involves designing and implementing an image stitching pipeline that is robust yet has low latency. The input will be a series of images taken by a drone, and the output will be a single high-resolution panorama image.

An envisioned usage scenario would involve drones capturing video footage over an area impacted by some disaster. This footage is then to be transmitted to a server for stitching. Using the stitched images, a clearer representation of the impacted area should

Figure 1: Example of stitched image consisting of six individual images. Some visual artifacts such as blurring might remain in the final panorama.

be visible, in case detecting persons in need of help or producing a clearer picture of the area was necessary.

4 Existing Literature

The original paper on image stitching [5] introduces a pipeline with several steps: feature matching, image matching, bundle adjustment, panorama straightening and blending. Many individual projects implement this pipeline and even some professional projects. OpenCV's image stitching module utilizes this pipeline to stitch multiple images together.

Next is a brief description of the pipeline according to the original paper:

1. **Feature Matching**: In this stage we find the regions of interest in each image of the sequence. A region of interest, commonly called a feature, could represent an area of the image with large variations in pixel intensities. Once a feature is found, relevant information about it is stored in a k-d tree. This k-d tree will store the points by their location in the image, so lookups of physically nearby features can be done in $O(\log(n))$ time [6].

2. **Image Matching**: In this stage we find connected sets of images which will be stitched together. To find appropriate pairs of images, the RANSAC algorithm is used. The algorithm will look at the probability that a certain match was generated by a correct image match.

3. **Bundle Adjustment**: In the previous step, matching pairs of images had been calculated based on inlier features. In this step, the bundle adjuster structure will adjust the camera parameters of each image added. This avoids cumulative errors resulting from the homography estimation. When each subsequent image is added to the bundle, for each feature the error is calculated from that feature in the other images.

4. **Panorama Straightening**: At this point, the image's parameters have been adjusted relative to each other;however in this step the the global camera is taken into account. Since the sequence of pictures are probably not taken along a perfectly even plane, the null vector of the covariance camera matrix is used to ensure the images are all taken in a single plane.

5. **Blending**: Given a panorama picture, the last step involves softening the image edges. This is done by assigning a weight function to the overlapping regions, which depends on their position from the edge of the image. Thus there is a more gradual change in the pixel values from one image to the next.

Since its publication this pipeline has been implemented in many open source alternatives, such as OpenCV and many smaller projects.

However, recently there have been newer methods that focus on improving certain parts of the pipeline. For example, the SuperGlue pretrained network [7] utilizes a neural network to perform feature matching and image matching.

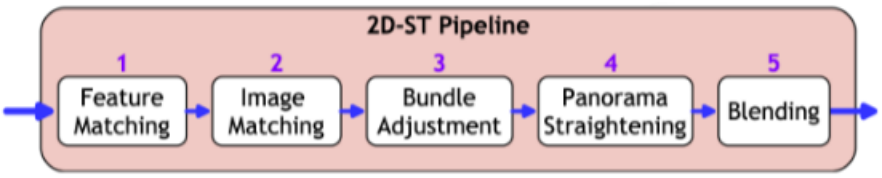

Figure 2: 2-D stitching pipeline based on original paper. The input consists of a set of images and the output consists of a single, high-resolution image.

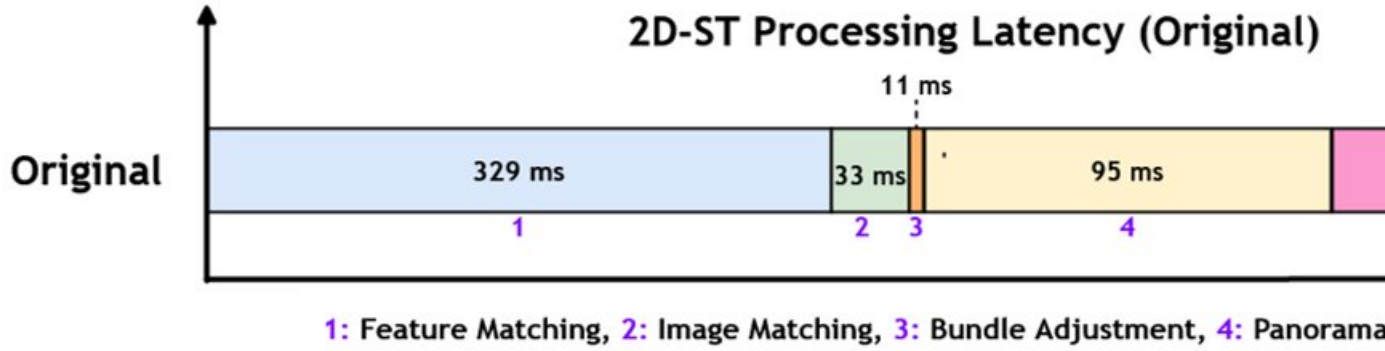

Figure 3: Timing diagram of first image stitching pipeline.

5 Resolution Method

The current project took the pipeline in Section 4 as a starting point. After understanding the pipeline, its inputs and outputs at each step, and some possible ways of implementing it in code, it was clear that some improvements could be made. For our application, the main focus was balancing the visual results with the latency. One of the first steps in the procedure was timing each step in the pipeline to identify possible areas of improvement.

The program used to time the pipeline utilized OpenCV, since it offers variety of the funcitons and algorithms we need to implement the pipeline step by step. Initially, two images were tested to determine the specific place where our program spent the longest amount of time. From Figure 5, it was clear the feature matching and blending.

This finding prompted a search for methods to improve on the latency of the pipeline for our specific use case. Among them was SuperGlue pretrained network, which uses a neural network to perform image matching and feature matching. The reason this project was chosen as a substitute for our original OpenCV implementation was due to its drastic improvement in terms of latency. In our original implementation, all the image processing ocurred at the image's native resolution and in full color. SuperGlue downsized the images to 640x480 resolution down from the iamge's native 1920x1080 resolution. Besides the downsampling, the images were also turned to greyscale for quicker processing of image data.

With only some slight changes to the source code, this project was integrated into the rest of the pipeline. Since the rest of our project was based around the OpenCV implementation, the programs output had to be compatible with the rest of the pipeline. Fortunately, this was not a problem since the model's output was of type `numpy.ndarray` which could easily be turned to `cv2.KeyPoint()` as was needed in the second step of the pipeline.

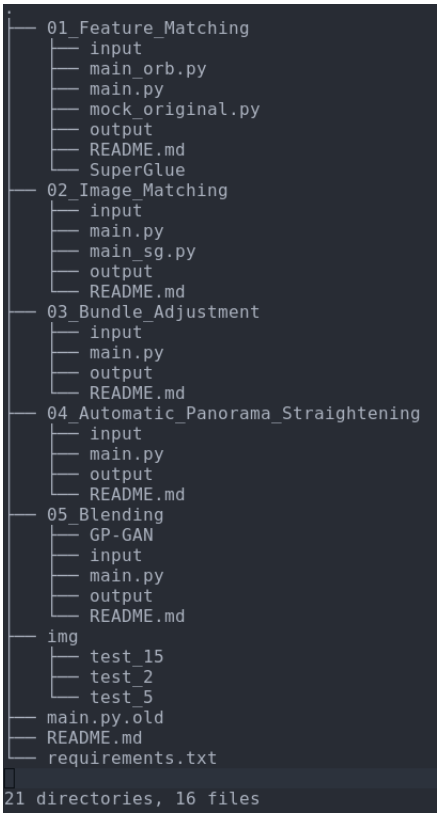

Figure 4: Main directory structure. There is a directory for each stage, with an `input` and `output` subdirectories where associated files are read from and stored to.

As a next step, an alternative blending method could also be researched. In the years since, there likely is a more efficient way of performing the blending process.

6 System Design and Implementation

The technical aspect of the project involved creating the pipeline step by step so that it could handle multiple images at once. Early on, the decision was made to split the pipeline into small scripts, each performing one step. Each script would read some files as inputs, perform its intended step, and write the files to an output directory. This was done in order to have a more convenient modification experience. By implementing the pipeline as small modules rather than a single script, changing any single step of the pipeline in the future would be easier. This proved to be the case when modifying the original feature matching step with SuperGlue. The only concern was to ensure the output followed the same format the next step, image matching, was expecting.

7 Result Analysis

The timing performance was performed by running the same pair or set of images using both methods. First, we just were concerned only with the first step of the pipeline, the feature matching stage. The `input/` folder contains merely the images to be stitched. The normal `main.py` script will utilize the original pipeline design with SIFT to perform feature matching. The `SuperGlue/` directory contains the software project for the

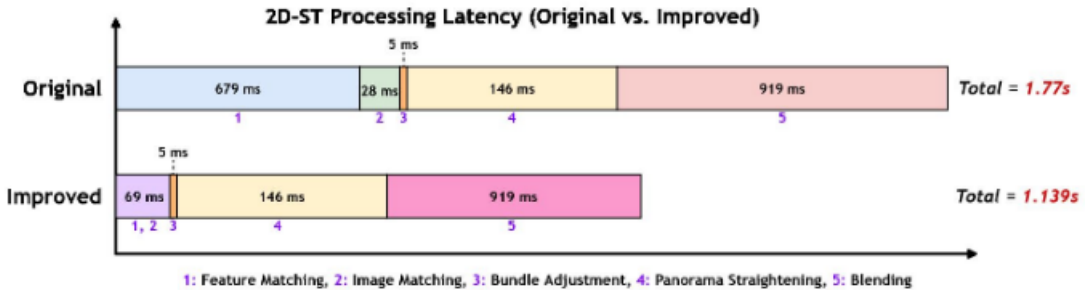

Figure 5: Timing results for the original SIFT-based feature matching method and the SuperGlue pretrained network.

alternative neural network method.

Based on Figure 7, the timing resulted in a faster implementation of the feature matching stage.

8 Conclusions and Contributions

References

[1] (Mar. 22, 2018). "Panorama," [Online]. Available: `https://www.britannica.com/art/panorama-visual-arts`.

[2] O. H. Connection. (). "Cincinnati panorama of 1848," [Online]. Available: `https://www.ohiomemory.org/digital/collection/p267401coll36/id/4168`.

[3] T. Cheredar. (2012). "The iphone 5's badass camera: 40% faster photo capture, panorama mode, & more," [Online]. Available: `https://venturebeat.com/2012/09/12/iphone-5-camera/` (visited on 09/12/2012).

[4] P. Toffanin, *OpenDroneMap: The Missing Guide*, first. UAV4GEO, 2019.

[5] M. Brown and D. G. Lowe, "Automatic panoramic image stitching using invariant features," *International journal of computer vision*, vol. 74, no. 1, pp. 59–73, 2007.

[6] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, Third. The MIT Press, 2009.

[7] P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich, "Superglue: Learning feature matching with graph neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2020.