Image Mosaicking for Low-Altitude Aerial Surveillance

Craig Western
Department of Mechanical Engineering
Stanford University
Stanford, CA USA
western.craig@gmail.com

Abstract—Overhead surveillance and mapping is becoming increasingly important in a number of areas with commercial, economic, and military applications. Effective algorithms for mosaicking are essential in building maps of large-scale areas; further, key steps in the mosaicking process such as feature matching and verification have implications in the rapidly growing area of robotic navigation and localization. This paper discusses an algorithm for the mosaicking of a set of images captured by a low-altitude unmanned aerial vehicle and a method of incorporating position metadata into the feature matching process to minimize the image search space and to minimize computation time. Implementation of the homography model with weighted blending between adjacent images produces a nicely mosaicked result with smooth transitions among images of the region of interest, selected to be the Stanford Oval. It is shown that searching for matching frames within a limited radius $R$ both saves computation time and improves likelihood of converging on a successful model.

Keywords—image stitching; mosaicking; weighted blending; unmanned aerial vehicle

I. MOTIVATION

With recent advances in and early proliferation of unmanned aerial vehicle (UAV) technology, overhead surveillance and mapping is finding a growing number of uses in topography, agriculture, resource management, emergency response, and military applications [1]. Mosaicking of multiple images captured at low altitude when building maps for such applications maximizes resolution and detail of the region of interest. In reconstructing the entire region, registering adjacent frames and minimizing seams and artifacts given variation in images due to vehicle orientation, lighting, and terrain become key questions in practical algorithms for low-altitude aerial surveillance. Previous work has investigated algorithms for overall image gain adjustment and blending across seams, implementing such techniques as weighted blending and multi-banded blending [2, 3].

Keypoint detection and frame registration also have implications in robotic navigation and localization. However, working with large datasets limits the computational efficiency needed to calculate position using images captured in real-time. Recent work has targeted the integration of position metadata into mosaicking algorithms, allowing computational speedups when the search space for potential matches is limited to a particular area rather than the entire image dataset [3, 4]. In particular, Yahyanejad studies this method for implementation in small-scale UAV navigation [5]. Current work in the Aerospace Robotics Lab at Stanford University seeks to integrate image sensing into navigation algorithms for autonomous underwater vehicles (AUVs) to reduce uncertainty in localization. Localization methods currently in place are based largely on terrain-relative navigation, which requires a benthic surface with defined terrain features and can be subject to error in flat areas [6]. Implementation of image-based localization, which is not dependent on elevation of the benthic surface, offers a mode of drift correction in areas of flat terrain.

Fig. 1. Stanford University (Courtesy Google Images)

Fig. 2. Global Hawk UAV (Courtesy Northrop Grumman)
The work discussed here aims to begin work in designing and verifying methods for localization of an aerial test platform with the long-term goal of implementing similar methods in AUV real-time localization algorithms. As first steps involve successful image registration, the goal of this project is to develop an algorithm to automatically register, mosaic, and blend images gathered by a low-altitude UAV over a small area of the Stanford campus and to integrate position metadata into the process of matching frames.

II. EQUIPMENT AND SETUP

The test platform chosen for image acquisition was a GPS-equipped 3DRobotics quadrotor running open-source ArduCopter software for navigation and control, shown in Fig. 3. A downward-facing GoPro HERO2 camera, shown in Fig. 4, was mounted to the underside of the frame and set to capture 5-megapixel images at fixed intervals of either 1.00 or 2.00 seconds during flyover, depending on test conditions.

Included in the hardware platform was a global positioning system (GPS) accurate to within a two- to three-meter radius, which proved to be sufficient for purposes of gathering the required position information. Among other sensors, an onboard barometer was used to record time-stamped relative altitude information. While the acquisition time of each image was not recorded directly, takeoff time was pinpointed using altitude information, matched with an image number by pinpointing takeoff in image space, and extrapolated to yield image acquisition times with roughly 0.5-second resolution. These times were then correlated with on-board GPS information to yield position in longitude and latitude for each acquired image.

The region chosen for mosaicking was a roughly 20 meter by 20 meter area in the center of the Stanford Oval, as the area was generally flat with distinct features and offered few objects that failed to fit standard affine and perspective models for image transformation (i.e., few objects not at ground level that would yield artifacts when seen from multiple angles). Using functionality included with the ArduPilot software, the quadrotor was directed to autonomously navigate through waypoints that resulted in a “lawnmower pattern” with 4 to 6 passes, allowing image acquisition over the entire region. Altitude varied from 7 meters to 11 meters depending on test conditions. Overlapping images stamped with XY-coordinate information allowed the generation of full mosaics and integration of position using the algorithm described below.

III. MOSAICKING ALGORITHM

The algorithm implemented for mosaicking is shown in Fig. 5 and described in detail below. Input information was limited to tiles for mosaicking, the ArduCopter data log containing GPS information, and camera calibration parameters.

A. Correction for GoPro Fish-Eye Distortion

An image distortion operation was first carried out on each input image in order to correct for the fish-eye effect of the GoPro lens. Camera calibration parameters were calculated using the Caltech Camera Calibration Toolbox for MATLAB, which follows a camera model similar to that used by Heikkil and Silven [7] that is described in detail in the documentation [8]. Parameters obtained by the procedure are the focal length in pixels (2x1 vector), the principal point coordinates in pixels (2x1 vector), the skew coefficient (scalar), and the distortion coefficients (5x1 vector), all specified in Table 1. Images were un-distorted using parameters extracted from the calibration.

<table>
<thead>
<tr>
<th>TABLE 1. Camera Calibration Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focal Length (Pixels)</td>
</tr>
<tr>
<td>Principal Point (Pixels)</td>
</tr>
<tr>
<td>Skew Coefficient</td>
</tr>
<tr>
<td>Distortion Coefficients</td>
</tr>
</tbody>
</table>
B. Scale-Invariant Feature Transform (SIFT)

Before implementing feature identification using SIFT, images were sub-sampled to a resolution of 480x640 pixels in order to limit the size of the input data. Additionally, because the quadrotor executed a half-turn for each pass in order to maintain forward motion, images captured every second pass were rotated by 180°. Doing so facilitated initial image alignment and improved convergence rate in the optimization discussed below. After adaptive histogram equalization to emphasize features, SIFT was performed on the grayscale images. The version selected for implementation, developed by VLFeat, allows the specification of parameters in order to modify the search process and improve results; specifically, a peak threshold of 1.7 was specified in order to eliminate features whose peaks were too small and to improve runtime with no significant loss in information. Both feature locations and descriptors were calculated.

C. Feature Matching Within Specified Radius R

After extracting SIFT features and descriptors from a particular image, a matching operation was carried out for all features in images that had already been processed, up to the current image. The procedure was only carried out, however, in cases where candidate matches lay within a certain radius of the current image. As described in Section II, position metadata was extracted from the flight data logs and correlated with each image taken during flight. The parameter R varied from 800 to 1000 depending on test conditions and was compared with the L2 norm of the difference in input longitudes and latitudes. R was chosen to be slightly larger than the maximum distance between overlapping frames. For each feature descriptor in a candidate image, the VLFeat feature matching algorithm found the closest descriptor in the neighboring image by finding the smallest L2 norm of the difference between features in question and all feature descriptors in the neighboring image.

D. Affine Pairwise RANSAC

Once distance between the current frame and a candidate match was found to be acceptable and feature matching was carried out, RANSAC (Random Sample Consensus) was implemented for the pair of images. For this step, the transformation was assumed to be affine to improve computation time, as the final model was found in the following error minimization step. Only frame matching information and resulting inliers were passed to the following step in order to drive the error minimization, and the affine assumption proved adequate to find candidate inliers.

Key parameters in the RANSAC implementation included selection of k = 3 correspondences per iteration for the affine assumption; iterating a conservative S = 500 times; and keeping inliers that fall within error E = 10 pixels, also a fairly high threshold. Maintaining high thresholds did not significantly contribute to computation time and helped to account for the mismatch in models between this step and the following step.

E. Error Minimization for All Frames

Using inliers for each pair of frames obtained from the RANSAC procedure, bundle error minimization over the entire dataset was carried out for matching frames. This step was taken in order to avoid the accumulation of error due to sequential image registration. By taking inliers produced by RANSAC and minimizing error over all matching frames relative to each other, chances of error due to bias in the images was less likely to result.

During the algorithm development process, error minimization was first carried out by making the affine assumption, which simplified calculations. When it was found that this model was inadequate as discussed in the following section, full homography was carried out. For the affine assumption, the problem was formulated as follows. For frames i and j, there is an affine transformation matrix \( A_i^j \) relating the frame i to the world frame and a transformation matrix \( A_j^i \) relating i to frame j, for which (if the frames are matched) there is a set of inliers \( M_i \) that can be obtained from inliers in the frame j through the equation

\[
M_i = A_i^j M_j .
\]  

The transformation matrix from the ground frame to the frame i can be obtained through the equation

\[
A_G^i = A_i^j A_j^G .
\]  

Using the equations above, the formula below can be obtained to relate inliers in frames i and j through the ground frame:

\[
A_j^G M_i = A_j^G M_j
\]  

Using all inliers among matched frames, the problem can be set to zero and formulated as a large linear least-squares problem

\[
0 = M_{full} P
\]  

In which \( M_{full} \) is a matrix populated by the pre- and post-transformation inliers, and \( P \) is a vector of length 6N that contains the 6 parameters specifying the affine transformation matrix \( A_j^G \) for the total number of frames N. The resulting \( P \) vector and matrices \( A_j^G \) allow placement of each frame relative to the ground frame. By formulating the equation as a least-squares problem, error among inliers can be minimized, yielding parameters \( P \) with minimal error accumulation due to bias.

For the full homography case, the problem can be formulated in a similar way. However, because equation (2) no longer holds in the perspective case, the bundle error minimization must be solved using a nonlinear optimization. In both cases, the solution of all transformation matrices allows image overlay with minimal error.

F. Image Overlay and Blending

Once a transformation matrix relative to ground for each image is obtained, images are simply multiplied by respective transformation matrices and placed on a canvas relative to the ground frame to build the mosaic. Doing so without implementation of a blending scheme causes obvious seams between images, however, due to slight misalignments and color imbalance. As such, two blending schemes were implemented with results discussed in the following section.
In the first blending scheme, pixels across the range of intersection between two images were multiplied by weights corresponding to their distance from the edge of the range of intersection. The scheme is illustrated in Fig. 6 for two images \( i \) and \( j \). In all cases, the width of the range of image overlap \( L \), in pixels, was taken to be the farthest horizontal distance (when images were overlaid horizontally) between two pixels in the area of overlap. The result was a smooth transition between images.

Using this method of blending, however, it was not straightforward to address the problem of arbitrary placement of an image on a preexisting canvas, which would not result in placement of images in a single direction in general. To account for this, a technique was adopted that was adapted from a weighted blending scheme referenced by Lowe [3]. In this method, a weight function

\[
W(x, y) = w(x)w(y)
\]

is assigned to each image where \( w(x) \) varies linearly from 1 at the center of the image to 0 at the edge. When overlaying a new image in an arbitrary position on the canvas, then, the function

\[
\frac{\sum_{x}^{L} I^i(x,y)w^i(x,y)}{\sum_{x}^{L} W^i(x,y)}
\]

determined the weight of each pixel \( i \) in the area of image intersection, where \( I \) is the value of the pixel and \( W \) is the value of the weighting function. This method led to smooth transitions through the region of intersection along with the ability to overlay an additional image at an arbitrary position. After applying the weighting function, the result was the full mosaic.

### IV. COMPARISON OF METHODS

The four methods described above were implemented, with key variables being the type of model (affine transformation versus perspective transformation) and method of blending. Results are discussed here, with illustrations in Figs. 7 to 10. All images were acquired at an altitude of approximately 7 meters, with subsampling to yield a tile resolution of 480x640 pixels and 9 tiles mosaicked together for each image.

Fig. 7 shows the results of assuming and implementing an affine transformation with no blending. Distortion and misalignment is apparent; looking in particular at the S and at the benches, there appear to be areas that do not align and do not fit the model. Artifacts from image overlay and ghosting are apparent, and there are clearly seams between images that might be mitigated by a blending scheme.

Fig. 8 illustrates the results of implementing the homography model after observing that the affine model did not allow images enough degrees of freedom to account for the mode of capture. In this case it can be seen that there is far less frame alignment, particularly along the edges of the S and near the benches. Further, the perspective distortion appears consistent, with images becoming wider to the right; it was noted that this is because of the quadrotor pitch angle during flight, which was slightly below zero during the image acquisition process as the quadrotor moved forward. With no blending, however, seams remain.

Fig. 9 shows the results of homography model implementation with linear blending as discussed in the previous section. The result is a smooth transition between frames; however, the method as described does not allow for the placement of a new frame in an arbitrary position on an existing canvas.
Fig. 10 illustrates the final implementation of homography and weighted blending, which does account for this issue. Transitions between frames remain smooth. Only pixels that fall within regions of image intersection are touched, and an image can be seamlessly combined with an existing canvas in an arbitrary position such that a seamless mosaic can easily be constructed. Some ghosting remains where features do not fit the homography model (e.g., near the benches), and more advanced techniques such as multi-banded blending or image merging along an edge would be required to suppress these artifacts.

V. RESULTS

Results of the full homography implementation with weighted blending across the entire Oval are shown in Fig. 11 and Fig. 12. Mosaics represent a total of 42 images, 17 in the upper portion of the Oval and 25 in the lower portion. Each image was captured at an altitude of 7 meters and subsampled to a resolution of 480x640 pixels. Drift occurred over the course of the quadrotor trajectory that prevented full overlap of the upper and lower images and was likely caused by wind or by error in following GPS waypoints; however, all tiles in the upper Oval and all tiles in the lower Oval were successfully mosaicked, with all tiles finding at least one corresponding match.

Fig. 12 shows a plot of frames and successful matches in longitude-latitude space for the Lower Oval. Because of the limited area of the region of interest and because there is no need for absolute location, absolute longitude and latitude is discarded, and coordinates are given in hundred-thousands of degrees relative to 37.429 degrees north latitude and -122.169 degrees west longitude. In the figure, plotted points represent frames overlaid on the canvas, with lines drawn between matched frames. It can be noted that plotted positions correspond to plotted images, with images more densely plotted near the left and a spreading out of images to the right. Because of the constant search space constraint R and higher likelihood of feature overlap, more matches were found where images occurred closer together.

An analysis of speedups was conducted for a single swath of the Lower Oval image shown above, for 11 images. With all other parameters identical, the frame matching search constraint R was set once to be 800 and once unimplemented (i.e., infinite). With the relatively tight constraining search parameter, the average execution time per iteration (including SIFT, feature matching, and affine pairwise RANSAC for each frame) was 4.065 seconds. With no constraint on the search, allowing possible feature matching across all images, the average execution time per iteration was 5.133 seconds. Choosing a parameter R comparable to the distance between frames saved 18.9% of the processing time, with all other parameters held constant. Additionally, failure to constrain the search to nearby frames actually caused the nonlinear optimization to converge on a bad model, as a false match was found on opposite sides of the swath. Here an additional benefit to constraining search parameters is highlighted, in that false matches between frames that are not neighbors are minimized.

VI. RELATED AND FUTURE WORK

While nearly seam-free mosaicking with minimal distortion was achieved, improvements upon the existing algorithm can be made. To further minimize ghosting in areas of high-frequency and in areas with objects that do not fit the perspective model, multi-banded blending could be implemented to join high frequencies and low frequencies across smaller and larger blending regions, respectively [2, 3]. Images could also be stitched at edges in order to largely eliminate ghosting due to image overlap. In the final results, some color imbalance can
also be noted. Lowe proposes a method of gain adjustment that could largely correct for this color imbalance [2], and implementing this procedure for gain adjustment would likely further improve the quality of the resulting mosaic.

In pursuing the use of image acquisition and vision in real-time localization, significant computational speedups must be made in order to carry out image processing on-board aerial or underwater test platforms. Trade-offs between image quality and process time should be investigated. In pursuing localization, a method should be chosen that retains the minimum amount of information needed to effectively assess the location of a moving platform; for example, the minimum number of SIFT keypoints needed to achieve image overlap should be investigated in order to improve on-board processing time. If accurate and seam-free image overlay is not needed, it may be appropriate to investigate the use of the affine model to allow solving by least-squares; orienting the camera such that an affine model is appropriate, or changing camera orientation according to robot position, may also allow the accurate use of such a model. Finally, lens optics change significantly when a camera is submerged. To pursue the application of such algorithms to underwater robotics, an appropriate model for underwater camera optics should be investigated and tested against the assumptions already discussed.

ACKNOWLEDGMENT

The author would like to thank the EE 368 staff for assistance through the duration of this work, along with the Aerospace Robotics Lab and the Monterey Bay Aquarium Research Institute for support in implementing the work conducted here.

REFERENCES

## APPENDIX: SUMMARY OF HOURS

<table>
<thead>
<tr>
<th>Date</th>
<th>Time Recorded</th>
<th>Total Hours</th>
<th>Tasks Completed</th>
</tr>
</thead>
<tbody>
<tr>
<td>5/7</td>
<td>9:00 – 11:00</td>
<td>2</td>
<td>MATLAB camera calibration</td>
</tr>
<tr>
<td>5/7</td>
<td>4:00 – 6:00</td>
<td>4</td>
<td>MATLAB camera calibration</td>
</tr>
<tr>
<td>5/8</td>
<td>10:00 – 10:30</td>
<td>4.5</td>
<td>Early mosaicking algorithms</td>
</tr>
<tr>
<td>5/26</td>
<td>9:00 – 11:00</td>
<td>6.5</td>
<td>Test mosaicking with predefined functions</td>
</tr>
<tr>
<td>5/27</td>
<td>8:00 – 1:00</td>
<td>11.5</td>
<td>Imaging and reading through existing code</td>
</tr>
<tr>
<td>5/27</td>
<td>9:00 – 11:00</td>
<td>13.5</td>
<td>Reading through existing code</td>
</tr>
<tr>
<td>5/28</td>
<td>7:00 – 9:00, 10:30 – 11:30</td>
<td>16.5</td>
<td>Imaging oval, learning about affine/ perspective trans.</td>
</tr>
<tr>
<td>5/29</td>
<td>10:00 – 12:00</td>
<td>18.5</td>
<td>Implementing new code from ARL</td>
</tr>
<tr>
<td>5/29</td>
<td>3:30 – 6:30</td>
<td>21.5</td>
<td>Developing ARL code</td>
</tr>
<tr>
<td>5/30</td>
<td>9:30 – 4:30</td>
<td>28.5</td>
<td>Color calibration, importing XY data, adapting code</td>
</tr>
<tr>
<td>5/31</td>
<td>10:30 – 1:30, 3:30 – 5:30</td>
<td>33.5</td>
<td>Met with Roland, implemented homography, boundaries</td>
</tr>
<tr>
<td>6/1</td>
<td>7:00 – 9:00, 10:30 – 5:30</td>
<td>42.5</td>
<td>Mosaic in 2D, debugging, thinking about blending</td>
</tr>
<tr>
<td>6/2</td>
<td>3:00 – 6:00, 7:30 – 11:30</td>
<td>49.5</td>
<td>Implementation of weighted blending, gain reading</td>
</tr>
<tr>
<td>6/3</td>
<td>11:00 – 3:00, 6:00 – 12:00</td>
<td>59.5</td>
<td>Meeting with Prof., poster</td>
</tr>
<tr>
<td>6/4</td>
<td>3:00 – 6:00, 9:30 – 10:30</td>
<td>63.5</td>
<td>Finalizing images, commenting code</td>
</tr>
<tr>
<td>6/5</td>
<td>10:00 – 11:00p</td>
<td>76.5</td>
<td>Working on report</td>
</tr>
</tbody>
</table>