Open Source Structure-from-Motion for Aerial Video

Matthew J. Leotta Eric Smith Matthew Dawkins Paul Tunison
Kitware, Inc. *
{matt.leotta, eric.smith, matt.dawkins, paul.tunison}@kitware.com

Abstract

Structure-from-motion (SfM) is a well-studied problem in the computer vision field and is of particular interest for aerial imaging applications like mapping, terrain modeling, crop monitoring, etc. With the current rapid growth in the commercial UAV and small satellite markets, aerial SfM is becoming even more important. In recent years, free and open source software has enabled almost anyone to apply SfM to their data at no cost. Existing free packages are oriented toward processing unordered collections of photographs and are less efficient at processing ordered collections or video. They are also prone to failure with nearly planar scenes that often arise in aerial photography. While commercial solutions for aerial SfM exist, they are proprietary and expensive.

This paper presents a new open source software toolkit named MAP-Tk that targets SfM for aerial video. It exploits temporal continuity and the aerial nature of the data to speed up and improve feature tracking, loop closure, and bundle adjustment. The system is highly configurable and modular. Dynamic plugins provide state-of-the art algorithms from other open source projects like OpenCV, VXL, and Ceres Solver. This paper presents the modular system design, user interface, novel algorithms exploiting aerial video, and results on various aerial data sets.

1. Introduction

The modern problem of structure-from-motion (SfM) in computer vision has its roots in aerial photogrammetry, a field that has been active for over one hundred and fifty years. Aerial photogrammetrists have manually used triangulation and bundle adjustment to build 3D maps of the world from photographs since the early days of photography from hot air balloons. Modern computer vision has greatly automated and advanced SfM over the last few decades by developing robust feature detection and matching techniques [2, 9] to automatically find correspondences across images and to do so over increasingly large sets of images [1, 4, 6]. Similarly, advances in non-linear optimization [20, 7, 23] have made large scale sparse bundle adjustment more feasible.

Figure 1. Aerial SfM: A UAV flies over a site and we wish to precisely estimate the camera pose at each image or video frame, a sparse 3D point cloud of observed landmarks, and optionally an orthorectified mosaic of the imagery.

In the last decade, the introduction of large scale photo sharing on the internet led to a trend in the computer vision community of studying SfM problems involving web-scale collections of unordered photographs. This body of work led to the development of the Bundler [19] software package, which is widely used today for SfM because it is free and open source. Also out of this movement came VisualSfM [24], another commonly used tool that makes SfM easy to run with a GUI and is free for non-commercial use.

We foresee that SfM research will follow the trends in data availability and soon make a return to its roots in aerial photogrammetry. The current boom in low cost com-

*The authors would like to thank AFRL Sensors Directorate for their support of this work via SBIR Contract FA8650-14-C-1820.

mercial and consumer UAVs is about to make aerial video more accessible than ever before. Likewise, upstart satellite providers like Skybox [18], Planet Labs [14], and UrtheCast [21] are paving the way for abundantly available imagery and video from space. SfM applied to aerial imagery can produce mosaics and 3D measurements while also localizing the sensor flight path, as shown in Figure 1.

In this paper we discuss the design of a new open source toolkit for SfM named MAP-Tk that is targeted specifically toward the challenges of aerial video. MAP-Tk is the **M**otion-imagery **A**erial **P**hotogrammetry **T**ool**k**it. Unlike prior open source SfM systems, our software is designed to be highly modular and flexible at runtime using a dynamic plug-in architecture described in Section 3. The goal is to allow algorithmic components to be easily interchanged at run-time for adaptability and ease of experimentation. The plugin framework also keeps required library dependencies to a minimum while allowing advanced algorithms from other toolkits like OpenCV [13], VXL [22], and Ceres Solver [3] to be added at run-time. Like VisualSfM, we provide a cross platform GUI application for visualization of the results. Unlike VisualSFM, our software, including the GUI, is fully open source with a permissive BSD licensed allowing for both commercial and academic use without copyleft constraints. The software is available now on GitHub[1] and community participation in its development is encouraged.

Section 4 explains how our system exploits the aerial nature of the data and approximate planarity of the ground through novel algorithms to speed up and improve feature tracking and loop closure. It exploits the temporal continuity of video to significantly speed up sparse bundle adjustment (SBA) by processing the data in a temporal hierarchy. The system can make use of GPS and IMU sensor data when available, but can also operate in the absence of other sensor data. While Bundler, VisualSfM, and similar systems can process aerial video frames as if they were unordered images, they do not exploit the temporal continuity and redundancy of video; this makes them inefficient. They also do not account for the often planer nature of aerial scenes; this occasionally makes them fail.

The system has been tested on aerial data ranging from a toy quadcopter video to military-collected aerial video (*e.g.* FMV and WAMI) to commercial satellite video from Skybox. Some of these results are discussed in Section 5.

2. Related Work

The primary SfM application addressed in the computer vision research community in the last decade has been 3D reconstruction from large-scale, crowd-sourced collections of tourist photographs. This application motivated the implementation of Bundler [19], the first open source package for end-to-end SfM on large image collections. Bundler was developed by Dr. Noah Snavely for his PhD dissertation. Bundler is a great application for unordered image collections, but it was not designed to be a configurable toolkit for adaptation to other applications, like video. Furthermore, the GPL license makes it less attractive for integration into closed source systems. While Bundler is still commonly used today, it is no longer actively being extended with new algorithms.

Another popular SfM tool is VisualSFM [24] by Dr. Changchang Wu. VisualSFM was also developed to support a PhD dissertation. Unlike Bundler, VisualSFM is not fully open source. Instead, the main application is closed source but "free for non-commercial use". Some of the underlying algorithms are also made available in open source libraries, again with GPL licenses. The two main advantages of VisualSFM over Bundler are that Visual SfM provides GPU acceleration for some algorithms and provides a GUI–based application with visualization of the results. VisualSFM is still targeted more at unordered image collections than aerial video. Yet the GUI does allow for some customization of the pipeline to make it more relevant to video. Like Bundler, VisualSfM is no longer actively developed.

OpenMVG [10] by Dr. Pierre Moulon, is yet another open source SfM toolkit, again built initially to support a PhD dissertation. OpenMVG uses the more permissive MPL license. It provides implementations of several recent algorithms in SfM but, like other packages, is aimed more at unorganized photos than aerial video.

While these packages are general enough to process aerial surveillance video, they are inefficient on video because they assume no order or consistency across the input images. On the other hand, there has been work targeting SfM efficiency in video. Shum *et al.* [17] proposed a hierarchical approach using key frames, and in recent work Resch *et al.* [15] presented numerous techniques aimed at optimizing SfM for video. Other work, such as Schweighofer and Pinz [16], addresses issues with SfM in planar scenes, which are common in aerial video. However, source codes for algorithms in these papers are not available. In contrast, our system provides reusable source code for solving these problems.

3. System Architecture

An important aspect of our software design is a highly modular and configurable architecture to allow for run-time switching between various algorithms at each stage of the processing pipeline. Furthermore, the modular design is needed to make it easy to plug in existing implementations of algorithms such as the feature detectors and descriptors available in open source libraries like OpenCV. Each of

[1] https://github.com/kitware/maptk

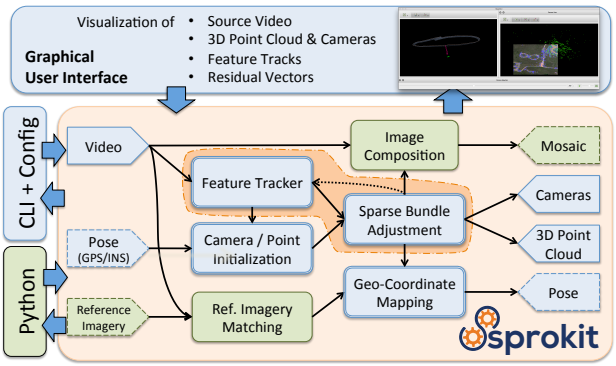

Figure 2. System architecture.

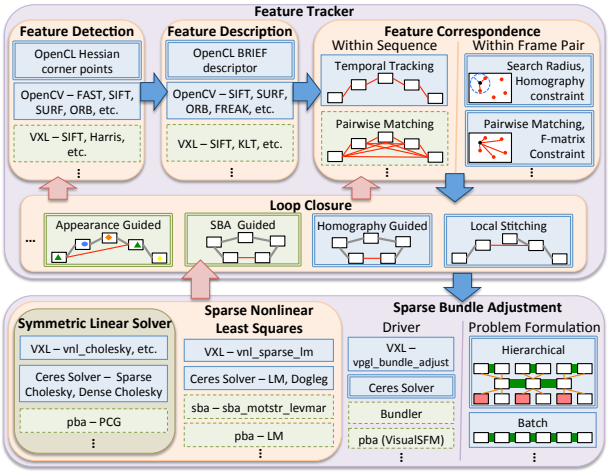

Figure 4. Detailed system architecture.

the key algorithms is interchangeable at run-time to provide various alternatives. The alternatives can be mixed and matched in any way and may be implemented using different third-party libraries. New algorithms can be plugged in easily without any changes to the core library and tools.

Figure 2 shows a flow diagram of our modular SfM pipeline. Rounded boxes illustrate the high-level components of the system with large blue arrows showing the flow of data between them. Blue components are those currently implemented while green components are either under development or on our roadmap. Data flow shown by black arrows is currently available in a batch processing mode through a combination of two command line interface (CLI) tools. The `maptk_track_features` tool runs the feature tracking algorithm on the video and writes feature tracks to disk. The `maptk_bundle_adjust_tracks` tool reads tracks to initialize camera poses and landmarks (*i.e.* 3D points), optimize cameras and landmarks with sparse bundle adjustment, and map the solution to geo-coordinates using either ground control points or GPS metadata. Each CLI tool accepts a configuration file that controls data input and output files as well as which algorithms to run and what parameters to use for each algorithm. The configuration file allows the selection and configuration of sub-algorithms that may be nested several layers deep.

Although the command line is the primary interface, one can also build custom applications by directly using the provided C++ API. Language bindings for C and Python are also under development. MAP-Tk provides a desktop GUI application, shown in Figure 3, that allows users to load imagery and SfM results and visualize the data in various ways. In upcoming releases, the GUI will also directly run the algorithms. The APIs and executable (including the GUI) are designed to be cross-platform to compile and run on Windows, OS X, and various flavors of Linux.

The high-level algorithms shown in Figure 2 only scratch the surface of the modular design. The system also supports nested abstract algorithms. Some implementations of the algorithm concepts in Figure 2 are written as functions of other abstract components. To illustrate, Figure 4 takes an expanded look at the Feature Tracker and Sparse Bundle Adjustment components. A provided implementation of Feature Tracker uses a detect-and-match approach. This implementation contains abstract components for Feature Detection, Feature Description, Feature Correspondence, and Loop Closure. Each of these sub-components has multiple implementations that can be used interchangeably, and some implementations of these sub-components may contain further nested sub-components. For example, one Feature Correspondence algorithm uses homographies to constrain the matches and it relies on an abstract Homography Estimation component. The same abstract Homography Estimation component and its implementations can be reused elsewhere, such as in a Loop Closure algorithm that depends on homographies. The choice of concrete algorithm instantiation and its parameters can be the same or different in each use.

When configuring a processing pipeline one can select which Feature Detector to pair with which Feature Descriptor and so on, or one can replace the entire Feature Tracker algorithm with an alternate implementation at the top level. In Figure 4, examples of some of the the concrete implementations of these abstract algorithms are shown as rectangular boxes. Blue boxes are implementations provided now. Green boxes are examples of implementations that could be added in the future. Blue arrows show the current feed-forward data flow between the components. Red arrows show optional feedback paths.

Observe that in Figure 4 several of the different implementations refer to third-party libraries like OpenCV, VXL, or Ceres Solver. Rather than requiring dependency on a large set of libraries to build, we use a dynamically loaded plugin architecture. As shown in Figure 5, the main li-

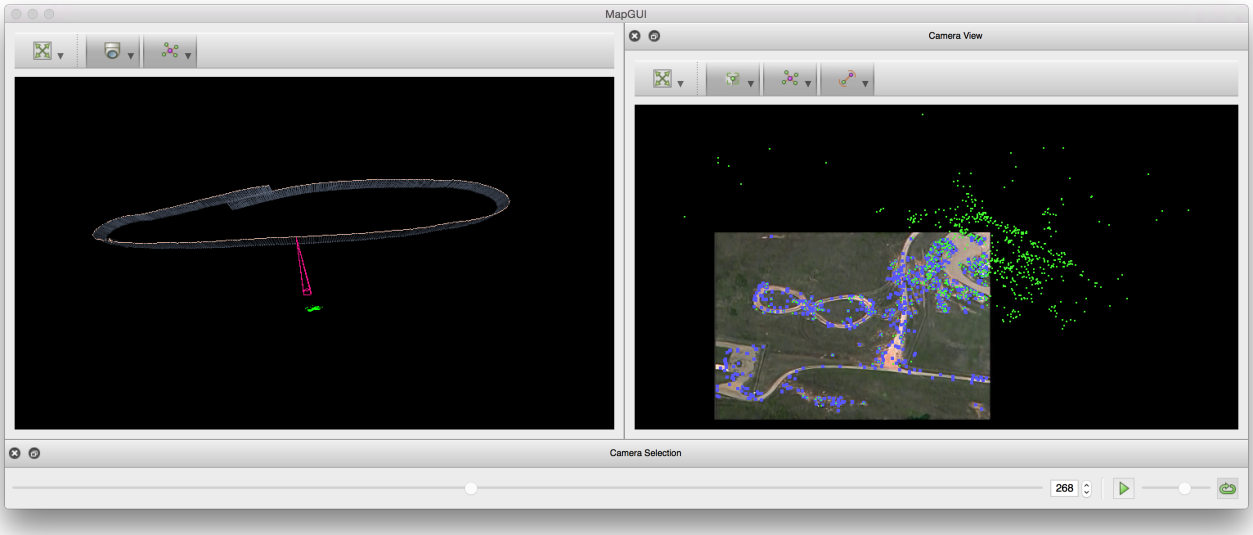

Figure 3. Screenshot of the GUI provided with the software. On the left is a 3D world view showing the path of camera (small grey frustums), the active camera (large magenta frustum), and 3D landmarks (green points). On the right is a 2D camera view showing the active image, tracked features (blue points), and projected landmarks (green points). A time slider at the bottom allows video playback.

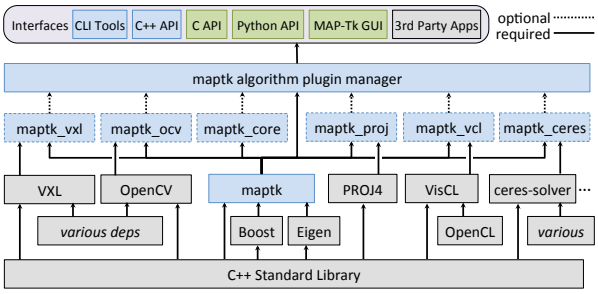

Figure 5. Library dependency diagram. The interfaces have only hard dependency on Eigen and Boost through the main library. Other libraries are optionally brought in at run-time via plugins.

brary depends only on Eigen (for linear algebra), a subset of Boost libraries (system, filesystem, timer) and the C++ standard library. The interfaces then only require these minimal dependencies to build. While the abstract algorithm interfaces are declared in the main library, their implementations are provided by various plugins that are loaded by a plugin manager at run-time. Each plugin may have additional dependencies.

The advantages of the modular strategy are two-fold. First, the user is not tied to any particular third-party libraries for algorithm implementations and can provide new implementations as desired. Second, the algorithms are interchangeable, so it becomes trivial to swap out various components of the system with alternatives and compare the results. Furthermore, our system provides the glue that allows interchange between various third-party algorithms that would otherwise use incompatible data structures. For example, an OpenCV feature detector can be run on an image loaded with VXL. Our abstract containers allow data to be passed from algorithm to algorithm in the native third-party data structures and only converted on demand. In particular this design allows construction of pipelines with a mix of CPU and GPU algorithms such that data is transferred automatically between CPU and GPU memory but only transferred when needed.

4. Algorithms

In addition to the modular software architecture, the system also provides algorithms for SfM that are specialized for aerial video. This section will cover two such algorithms. The first algorithm describes an approach to feature tracking and loop closure in aerial video that exploits temporal continuity and the often-planar nature of the ground. The second algorithm describes a method for taking advantage of temporal continuity to speed up bundle adjustment.

4.1. Feature Tracking and Loop Closure

Feature tracking is the process of detecting salient points in one frame and finding the location of the same features when they appear in other frames. In one implementation, the system relies on common feature detection and matching algorithms from other libraries like OpenCV to provide this core capability. Feature tracking is a noisy process in practice. While many features are correctly tracked, a non-trivial amount of features are also mismatched. Furthermore, not all of the correct tracks belong to stationary features. Moving foreground objects will generate outlier

tracks that are not geometrically consistent with the model of a moving camera observing a static scene. These invalid and outlier tracks can create big problems for SfM if not properly filtered out. The usual approach to dealing with these outlier tracks is to estimate the fundamental matrix [5] (or essential matrix [11] for calibrated cameras) between the pair of images and use the fundamental matrix as a constraint to remove geometrically invalid matches.

Our approach uses a homography constraint instead of the fundamental matrix. The homography is a much stronger constraint, but is only valid for planar scenes or for fixed cameras that may rotate but do not translate. In aerial video, the frame-to-frame motion is usually small enough, and/or the ground is planar enough that this constraint holds reasonably well over nearby video frames. In fact, for adjacent frames of video, the fundamental matrix becomes very unstable for small camera motions and a homography is much more robust. Since we assume temporal continuity of the video, we can track features from frame-to-frame through the video using this homography constraint. Frame-to-frame tracking has the advantage of being $O(n)$ in the number of frames rather than consider all pairs of images, which is $O(n^2)$. Of course, only tracking features between adjacent frames of video is fragile and prone to failure if there is a single bad frame. It also does not account for features that become occluded or exit the field of view, but then are observed again at a later time. Normally, when a feature goes out of sight and then reappears, the track is broken and a new track forms. In some cases we can stitch these disjoint tracks of the same feature back together through the process of loop closure. Loop closure can significantly reduce drift in the SfM solution.

There are two types of loop closure currently integrated into MAP-Tk: local loop closure and homography-guided loop closure. Local loop closure is used to link tracks over a small number of bad frames with corrupted, noisy, or missing pixels. Homography-guided loop closure is used to link tracks that have left the field of view due to camera motion and then later returned. These two loop closure approaches are complementary and can be used together for best results.

Our local loop closure approach estimates homographies between adjacent frames as is done in video stabilization. Occasionally a bad frame (or sequence of bad frames) causes frame-to-frame feature tracking to fail, which splits the video results into a series of stabilized "shots". The algorithm monitors the number of tracks maintained from frame-to-frame. When the fraction of tracked feature drops dramatically the current shot ends. When the the number of successful tracks returns to previous good levels, a new shot starts. After a new shot has started, the algorithm performs additional feature matching between the current good frame and good frames at the end of the last shot to try to recover lost tracks.

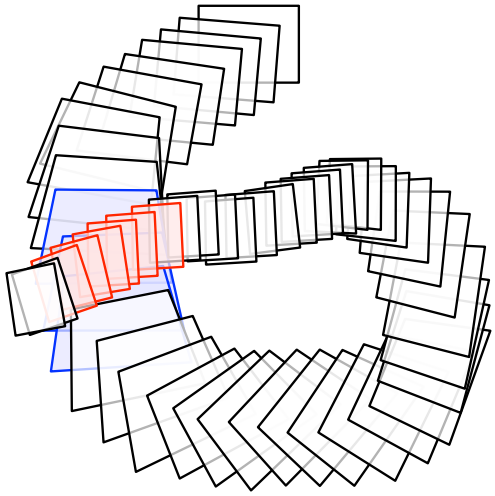

Figure 6. Loop closure. Homographies stabilize frames into common coordinates. Tracks in red frames should be matched against the blue frames to link duplicate tracks and close loops.

Our homography-guided loop closure algorithm addresses the long-term loop closure problem and is also related to aerial video stabilization. This algorithm is built on the assumption that aerial video views a roughly planar ground and homographies can be used to register the frames to a common plane over a long time period. This assumption does not hold for all aerial video, but generally holds when the distance of the camera to the ground is large compare to the height of objects off the ground. The algorithm uses the estimated homographies from feature tracking to predict pairs of frames to be matched for loop closure. In video stabilization, one estimates the sequence of homographies that can map all frames into a common coordinate system. As shown in Figure 6, we can identify when the sensor has returned to looking at a part of the scene it has seen before by looking at frame boundary intersection in this registered space. In the figure, tracks in red frames should be matched against the blue frames to link duplicate tracks and close loops. The homographies will contain drift over time, but our results show that, for aerial video, the homographies are accurate enough to identify approximate frames (or frame ranges) to search for loop closure. This approach is very simple and can run online with minimal overhead added to normal frame-to-frame tracking. In practice, our implementation is accelerated further by selecting a key frame whenever the camera motion is such that frame overlaps with the last key frame drops below a threshold. In loop closure, we then only need to match against the selected key frames.

4.2. Hierarchical Sparse Bundle Adjustment

Given a set of images and feature tracks, there are multiple ways to formulate a SfM strategy. By *strategy* we mean

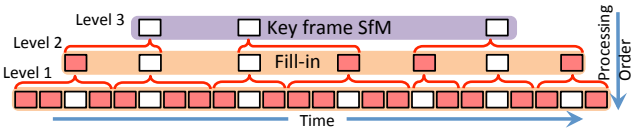

Figure 7. Temporally hierarchical structure-from-motion using key frames to reduce the size of the bundle adjustment problem.

determining how to initialize the model parameters, which parameters to optimize, how to exploit sparsity, and potentially how to factor the problem into smaller sub-problems.

Our SfM strategy is to use a temporally hierarchical approach related to the work of Shum *et al*. [17]. The approach is diagrammed in Figure 7 for the case of three levels, but as few as two levels may be sufficient. We first subsample the video frames temporally to produce a much smaller sub-problem with less redundancy between frames. If the key frames are selected appropriately, a sparse bundle adjustment on this sub-problem will produce cameras and landmarks with a similar solution for those structures to that of the original problem. After completing the key frame bundle adjustment, a second fill-in stage can efficiently add back in cameras that were skipped by interpolating their parameters from neighboring key frame cameras and optimizing each added camera independently while keeping the landmark locations fixed. The fill-in SfM can conclude with a final bundle adjustment of all cameras, but the solution is usually good enough that a final SBA step is not needed. Applying SBA with this hierarchical strategy can reduce bundle adjustment times by orders of magnitude.

The appropriate selection of key frames is an important factor in achieving speed-up while still retaining a correct and accurate solution. Our initial experiments have used a fixed sampling rate (e.g. every nth frame). This strategy work wells on videos with smooth and constant speed camera motion. However, the best key frame rate depends on many factors including video frame rate and the speed of the platform motion. A fixed sampling rate does not work well for more erratically moving cameras. In that case an algorithm would be needed to dynamically adjust the sampling rate based on feature tracking statistics.

5. Experimental Results

We have evaluated our system on various types of aerial video. These include publicly released aerial video, military aerial video, commercial satellite video, and hobbyist quadcopter video. The video type ranges from FMV (high frame rate, HD or less resolution) to WAMI (very high resolution, lower frame rate). Each video has different flight parameters, but fixed stare point missions (*e.g*. site monitoring) are the most common in our data and best suited for SfM. The following is a list of the data sets evaluated. Results and discussion on the first three of these datasets are provided in following subsections.

- CLIF 2007 [8] – public WAMI data from AFRL; only one of six cameras is used; orbiting flight, fixed stare point outside of image bounds
- MAMI-I – WAMI-like data from AFRL (not publicly available); orbiting flight; stare point in image
- VIRAT Video Dataset [12] – public FMV-like data; orbiting flight; stare point in image
- Operational FMV – orbiting flight; stare point in image
- Skybox – satellite video from Skybox/Google; mostly linear platform motion (earth orbit), stare point in image.
- Toy Quadcopter – Low altitude video from a toy quadcopter in a backyard; erratic flight

The CLIF 2007 data nicely complements the VIRAT and MAMI datasets and together they cover two different extremes that are likely to be encountered in aerial video SfM. The CLIF data, when limited to just one camera, has the properties of a fast roaming mode of operation. The field of view is continuously sweeping across the ground. Tracked features come into the field of view, move quickly across the image with large frame-to-frame motion, and then exit the field of view typically within about 10 frames of video. The orbiting pattern in CLIF results in the field of view sweeping back over the same locations hundreds of frames later, providing a good test bed for long-term loop closure approaches.

The VIRAT, MAMI-I, and operational FMV data have very different properties. The stare point is fixed near the center of the field of view and the frame rate is relatively high. The results is that feature points move slowly across the image and stay in the field of view for hundreds of frames. The stare point in the center of the field of view makes long-term loop closure less critical on this data.

Skybox video also has a relatively fixed stare point, but the camera path is closer to linear and the field of view is very narrow. Conversely, video from the toy quadcopter has a wide field of view and very erratic camera motion and camera orientation.

Unless otherwise noted, our experiments use the SURF feature detector/descriptor and FLANN matching from OpenCV. The matching is augmented with a homography filter and loop closure is guided by homographies as in Sec. 4.1. Links to configuration files for public data sets are provided on the MAP-Tk Github site for reproducibility.

5.1. CLIF 2007

We evaluated the homography-guided loop closure on CLIF 2007 [8] imagery and found it to be quite successful. A single camera from CLIF 2007 provides a good test bed for loop closure because the camera sweeps across the

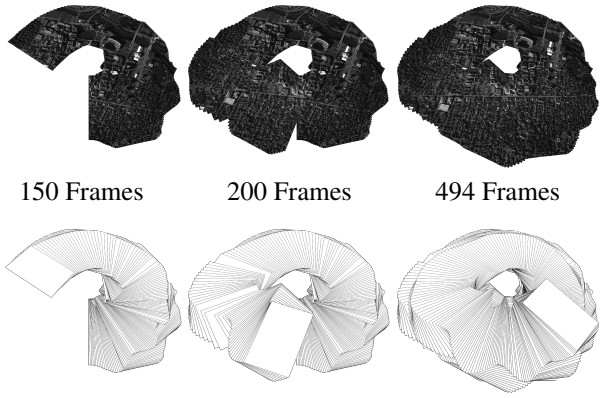

Figure 8. Homography-guided frame registration on CLIF 2007 data. Top: mosaic images. Bottom: frame outlines. After one complete orbit visual structures are still in approximate alignment.

ground in large arcs and revisits the same location several times.

Figure 8 shows examples of homography registration of the CLIF data into a common plane: the coordinate system of the first frame. The top row shows mosaic images generated from registering the imagery; the bottom row shows the borders of each individual video frames in the warped space. It is clear that there is only small drift after one complete orbit. For example, roadways spanning the boundary between the 1st and 200th frame line up well, but not perfectly. Drift accumulates further in the second orbit, but still by amounts smaller than the frame size. Therefore, we can use the overlap of the warped image boundaries to predict when to do loop closure. In our current implementation, enabling homography-guided loop closure adds about 7% to feature tracking time on the CLIF 2007 data. In CLIF 2007, this loop closure fires frequently after the first orbit because the second orbit follows a very similar path to the first and almost every frame presents a loop closure opportunity.

Figure 9 shows the results of bundle adjustment both with and without homography-guided loop closure. The impact is considerable. Without loop closure there is drift between orbits; cameras and 3D points each spiral out of their respective horizontal planes. With loop closure the camera's path and 3D point cloud are each more consistent and nearly lay in common planes, as expected. Furthermore, the reconstructed 3D point cloud is much closer to the ground plane at Z=0 (represented by a horizontal gray line in Figure 9 side views). Finally, with homography-guided loop closure enabled, SBA converged in 111 iterations rather than 257 iterations without loop closure. Thus, SBA with loop closure is about twice as fast. We should also note that the average projection error, measured as root mean square error (RMSE), is slightly higher (1.02 pixels) with loop closure than without (0.85 pixels). Error increase is due to observation of loop closed 3D points in more frames on average.

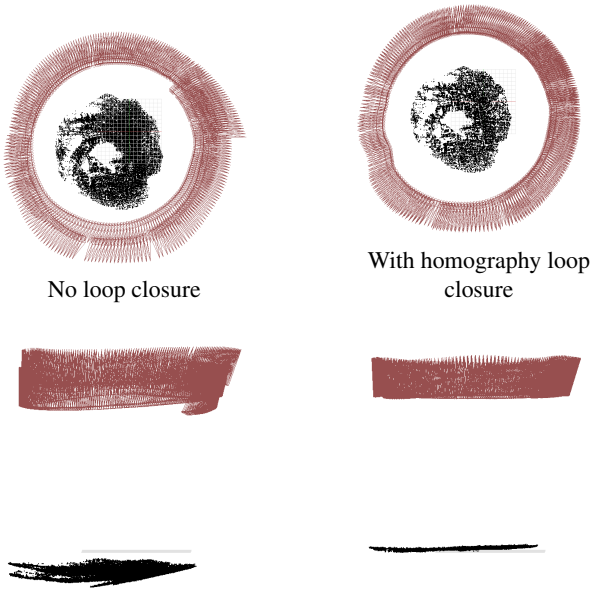

Figure 9. Results of SBA without loop closure (left) and with homography-guided loop closure (right). Cameras are shown as red frustums; 3D points are in black; ground plane at Z=0 shown as a gray line.

5.2. MAMI-I

In evaluating our hierarchical SBA approach, it was expected that a final full bundle adjustment at the original temporal scale would not be needed due to the amount of redundancy between adjacent frames. Our experiments on MAMI-I data found this hypothesis to be correct when the camera down-sampling rate was small enough. With the appropriate sub-sampling of both cameras and points, we were able to achieve one to two orders of magnitude speed improvements over standard batch SBA. One relevant experiment used 4494 frames of MAMI-I video starting with GPS/INS initialization and with 44949 feature tracks, each more than 50 frames long. Naively running batch SBA on this size problem took 28 iterations and 150291 seconds (41hr 44min 50 sec) to converge to a final root mean square error (RMSE) of 1.229 pixels. For comparison, hierarchical bundle adjustment was configured to start with every 50th frame (91 frames in total) and then fill in 3 frames at each level. The initial SBA on 91 frames took 32 iterations, but only 22 seconds. The second SBA on 361 frames took only 2 iterations and 18 seconds. The third SBA on 1441 frames took only 2 iterations and 428 seconds (7min 8sec). The final SBA on all 4494 frames took only 2 iterations and 10444 seconds (2hr 54min 4sec). The combined hierarchical SBA result is an order of magnitude faster than the baseline SBA result with a RMSE of 1.302 pixels – only 6% higher than the baseline. Results are visually identical. However, the RMSE for the 4494 frames before the last SBA was already

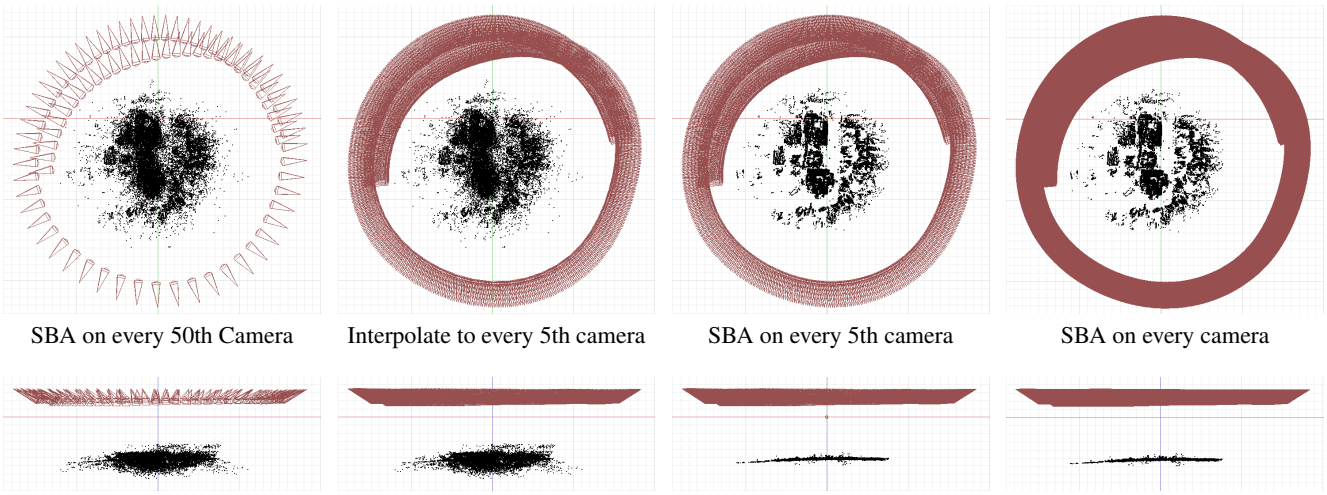

Figure 10. Hierarchical Sparse Bundle Adjustment on MAMI-I (sequences 152*). Feature tracks are selected that span at least 50 frames. From left to right: 1. SBA run on every 50th frame (91 cameras total); 2. interpolate to provide every 5th frame, (991 cameras total); 3. SBA on interpolated cameras produces almost no change in cameras; 4. Interpolated all missing cameras and final SBA, almost no change.

1.307 pixels. This means the final, SBA took nearly 3 hours to squeeze out the last 0.4% of RMSE. If we skip the final SBA step, the hierarchical approach is two orders of magnitude faster than the baseline. Figure 10 shows a similar hierarchical result on MAMI-I data.

5.3. VIRAT Aerial Public Data Set

The VIRAT aerial video public data set [12] has similar properties to operational FMV making it a good surrogate for algorithmic development in environments, like universities, where ITAR or classified data is not allowed. The limitations of the VIRAT data are that it is interlaced with lower resolution (480i), the camera motion is sometimes erratic, and the encoded metadata stream is unreliable. We ran MAP-Tk on clip 09172008flight1tape3_2 from the dataset. We ignored the metadata and applied deinterlacing as a preprocessing step. Figure 11 shows the results of MAP-Tk on 621 frames sampled from 09172008flight1tape3_2 at 2Hz. This result required 12.6 minutes for feature tracking and 5.6 minutes for SfM. The final root mean square reprojection error is 1.3 pixels.

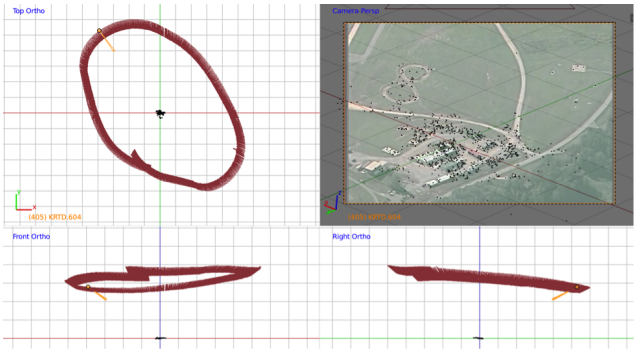

Figure 11. Result on video 09172008flight1tape3_2 from the VIRAT public data set [12]. Cameras and points shown from three principal directions as well as projected into an image.

6. Conclusions

We have presented MAP-Tk, a new open source toolkit for structure from motion on aerial video. The software design is highly modular and plugin-based to allow reconfiguration and experimentation with various different algorithmic components. Included in the toolkit is a set of algorithms specifically targeted at making structure-from-motion more efficient and reliable for aerial video. The algorithms exploit the temporal continuity of the data and planarity of the ground. We have evaluated the software on a range of different types of aerial video and demonstrated improvements in accuracy and speed.

We are continuing to develop MAP-Tk and plan to make the algorithms more automated and adaptable to variations in the data. For example, we will select key frames for hierarchical bundle adjustment in a way that is automatically driven by feature tracking statistics. We will also improve the GUI and C/Python interfaces for the software as well as adapt the pipeline to run in a multi-threaded streaming mode. Most importantly, we are building a community of users and contributors around this new open source framework. We hope that as the developer community grows, this software will provide an easy framework for evaluation of new algorithms and comparison to existing algorithms within the context of a complete SfM pipeline. At the same time, the permissive license and various system APIs it will provide an easy transition path for SfM research into practical applications.

References

[1] S. Agarwal, N. Snavely, I. Simon, S. Seitz, and R. Szeliski. Building rome in a day. In *IEEE International Conference on Computer Vision (ICCV)*, pages 72–79. IEEE, 2009. 1

[2] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *European Conference on Computer Vision (ECCV)*, pages 404–417. Springer, 2006. 1

[3] Ceres Solver – C++ sparse nonlinear least squares solver. http://code.google.com/p/ceres-solver/. 2

[4] J. Frahm, P. Fite-Georgel, D. Gallup, T. Johnson, R. Raguram, C. Wu, Y. Jen, E. Dunn, B. Clipp, S. Lazebnik, et al. Building rome on a cloudless day. *European Conference on Computer Vision (ECCV)*, pages 368–381, 2010. 1

[5] R. Hartley and A. Zisserman. *Multiple view geometry in computer vision*, volume 2. Cambridge Univ Press, 2000. 5

[6] J. Heinly, J. L. Schonberger, E. Dunn, and J.-M. Frahm. Reconstructing the world* in six days *(as captured by the yahoo 100 million image dataset). In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015. 1

[7] Y. Jeong, D. Nister, D. Steedly, R. Szeliski, and I. Kweon. Pushing the envelope of modern methods for bundle adjustment. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1474–1481. IEEE, 2010. 1

[8] A. F. R. Laboratories. Columbus large image format (clif) 2007 dataset. https://www.sdms.afrl.af.mil/index.php?collection=clif2007. 6

[9] D. G. Lowe. Object recognition from local scale-invariant features. In *IEEE International Conference on Computer Vision (ICCV)*, volume 2, pages 1150–1157. Ieee, 1999. 1

[10] P. Moulon, P. Monasse, R. Marlet, and Others. OpenMVG – an open multiple view geometry library. https://github.com/openMVG/openMVG. 2

[11] D. Nistér. An efficient solution to the five-point relative pose problem. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(6):756–770, 2004. 5

[12] S. Oh, A. Hoogs, A. Perera, N. Cuntoor, C.-C. Chen, J. T. Lee, S. Mukherjee, J. Aggarwal, H. Lee, L. Davis, et al. A large-scale benchmark dataset for event recognition in surveillance video. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3153–3160, 2011. 6, 8

[13] OpenCV – open source libraries for computer vision. http://opencv.org/. 2

[14] Planet Labs. https://www.planet.com/. 2

[15] B. Resch, H. P. A. Lensch, O. Wang, M. Pollefeys, and A. Sorkine-Hornung. Scalable structure from motion for densely sampled videos. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015. 2

[16] G. Schweighofer and A. Pinz. Robust pose estimation from a planar target. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(12):2024–2030, 2006. 2

[17] H. Shum, Q. Ke, and Z. Zhang. Efficient bundle adjustment with virtual key frames: A hierarchical approach to multi-frame structure from motion. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2. IEEE, 1999. 2, 6

[18] Skybox Imaging. http://www.skyboximaging.com/. 2

[19] N. Snavely and Others. Bundler - structure from motion (sfm) for unordered image collections. http://www.cs.cornell.edu/~snavely/bundler/. 1, 2

[20] B. Triggs, P. F. Mclauchlan, R. I. Hartley, and A. W. Fitzgibbon. *Bundle Adjustment – A Modern Synthesis*, volume 1883. January 2000. 1

[21] UrtheCast. https://www.urthecast.com/. 2

[22] VXL – C++ vision libraries. http://vxl.sourceforge.net/. 2

[23] C. Wu, S. Agarwal, B. Curless, and S. M. Seitz. Multicore bundle adjustment. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3057–3064. IEEE, 2011. 1

[24] C. Wu and Others. Visualsfm - a visual structure from motion system. http://homes.cs.washington.edu/~ccwu/vsfm/. 1, 2