IMPLEMENTING FUNCTIONAL MODULARITY FOR PROCESSING OF GENERAL PHOTOGRAMMETRIC DATA WITH THE DAMPED BUNDLE ADJUSTMENT TOOLBOX (DBAT)

N. Börlin[1,*], A. Murtiyoso[2], P. Grussenmeyer[2]

[1] Department of Computing Science, Umeå University, Sweden — niclas.borlin@cs.umu.se
[2] Photogrammetry and Geomatics Group, ICube Laboratory UMR 7357, INSA Strasbourg, France — (arnadi.murtiyoso, pierre.grussenmeyer)@insa-strasbourg.fr

Commission II

KEY WORDS: Open-source software, bundle adjustment, flexible computations, photogrammetry

ABSTRACT:

The Damped Bundle Adjustment Toolbox (DBAT) is a free, open-source, toolbox for bundle adjustment. The purpose of DBAT is to provide an independent, open-source toolkit for statistically rigorous bundle adjustment computations. The capabilities include bundle adjustment, network analysis, point filtering, forward intersection, spatial intersection, plotting functions, and computations of quality indicators such as posterior covariance estimates and parameter correlations. DBAT is written in the high-level Matlab language and includes several processing example files. The input formats have so far been restricted to PhotoModeler export files and Photoscan (Metashape) native files. Fine-tuning of the processing has so far required knowledge of the Matlab language.
This paper describes the development of a scripting language based on the XML (eXtensible Markup Language) language that allow the user a fine-grained control over what operations are applied to the input data, while keeping the needed programming skills at a minimum. Furthermore, the scripting language allows a wide range of input formats. Additionally, the XML format allows simple extension of the script file format both in terms of adding new operations, file formats, or adding parameters to existing operations. Overall, the script files will in principle allow DBAT to process any kind of photogrammetric input and should extend the usability of DBAT as a scientific and teaching tool for photogrammetric computations.

1. INTRODUCTION

1.1 Background

Bundle adjustment is a crucial part of photogrammetry and computer vision. Several commercial photogrammetric software include a dedicated module for the computation of bundle adjustment, e.g., Trimble Inpho and ERDAS (Lumban-Gaol et al., 2018). In other software, the bundle adjustment might be hidden in a more black-box manner. Open source solutions to the bundle adjustment problem also exist in the form of toolboxes and/or dedicated functions. Examples include the Apero module from the Apero-MicMac photogrammetric suite (Rupnik et al., 2017) or the DGAP developed by the University of Stuttgart (Cramer, 2006). From the computer vision domain, other open source options also exist, for example Sparse Bundle Adjustment (SBA) (Lourakis, Argyros, 2009), multi-core bundle adjustment (MCBA) (Wu et al., 2011), or the Bundler software (Snavely et al., 2008). Most of these solutions are written in a low-level language such as C or C++.

The Damped Bundle Adjustment Toolbox in Matlab (DBAT) is written in the high-level Matlab language (Börlin, Grussenmeyer, 2013a). One goal is to provide open-source, easy-to-use implementations of core photogrammetric computation, including computation of posterior statistical quality estimates.

Several demo examples and data sets are included in DBAT. However, unless only minor modifications of the examples are needed to suit the user needs, programming skills have been required to use DBAT on the user's own data. Furthermore, the user has so far been restricted to either Photomodeler or Metashape (Photoscan) formats as input.

1.2 Aim

The aim of this paper is to present a modular, script-based, processing of photogrammetric data. The goal is to reduce the level of needed programming skills and allow for a wider range of input data sources.

2. THE DAMPED BUNDLE ADJUSTMENT TOOLBOX (DBAT)

2.1 Background

As the name suggests, the Damped Bundled Adjustment Toolbox started out as a toolbox to study different damping strategies known in non-linear optimisation (Börlin, Grussenmeyer, 2013a, Börlin, Grussenmeyer, 2013b). The source code has been available on GitHub since February 2014 [1].

An early application was camera calibration (Börlin, Grussenmeyer, 2014). Later, the focus shifted into validating other, closed-source, Photogrammetric software, such as Photomodeler and Metashape (then Photoscan) (Börlin, Grussenmeyer, 2016, Murtiyoso et al., 2017, Murtiyoso et al., 2018).

A recent development was to modularise the bundle projection model to allow users to develop novel projection models for, e.g., non-standard lenses or underwater applications (Börlin et al., 2018, Menna et al., 2018, Börlin et al., 2019).

*Corresponding author

[1] https://github.com/niclasborlin/dbat

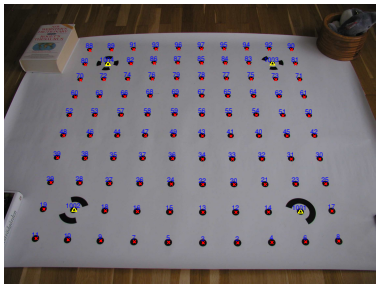

Figure 1: An input image with measured coordinates.

2.2 Current capabilities

The current DBAT release [2] includes the following capabilities:

2.2.1 Bundle adjustment The bundle adjustment proper, with or without self-calibration. Fixed and weighted prior observations are supported, e.g., control points and camera stations, as are check points. The parameters to be estimated by the bundle are selectable at the parameter level, e.g., for individual camera parameters. Furthermore, the parameters can be block-invariant (the same for a whole block), image-variant (individual for each image), or anything in between.

Multiple damping schemes may be used to avoid divergence due to poor initial values (Gauss-Newton-Armijo (default), classical Gauss-Markov, Levenberg-Marquardt, or Levenberg-Marquardt-Powell) (Börlin, Grussenmeyer, 2013a).

Posterior covariance estimates are computed from the bundle result, including correlations and significance levels, point and image quality statistics.

2.2.2 Photogrammetric processing Besides bundle adjustment, DBAT supports several other core photogrammetric computations, e.g., spatial resection, forward intersection, and absolute orientation.

2.2.3 Software compatibility DBAT can input data from Photomodeler-style text export files and point tables. Furthermore, DBAT can read and post-process native Metashape (Photoscan) `.psz` files.

2.2.4 Network analysis and quality DBAT includes functions to filter points and analyse camera networks to detect network problems. The filter allows removal of points with low point count and/or intersection angles. The analysis functions can detect both structural rank problems, e.g., caused by missing observations or gaps in the network, and numerical problems, e.g., caused by weak networks.

2.2.5 Output and report generating DBAT can present the results graphically in 2D and 3D and in text files. The available 2D plots present the raw data (Figure 1) or image and point quality statistics, such as image coverage, ray count, ray intersection angles, posterior standard deviations, etc., (figures 2–3). Other 2D plots illustrate the evolution of the bundle process (Figure 4). The 3D plots either show the final camera network or the evolution of the camera network and OP parameters during the bundle process (Figure 5).

The text output is mainly a PhotoModeler-style text summary report file that contain the processing input and setup, any detected problems, the estimated values and posterior covariances. Any detected high correlations among the estimated parameters are also listed.

[2] DBAT v0.8.5.1, released January 13, 2019.

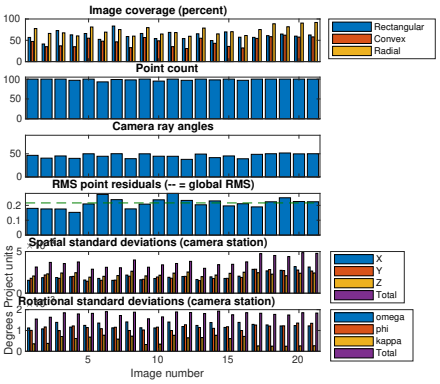

Figure 2: Image and external orientation statistics.

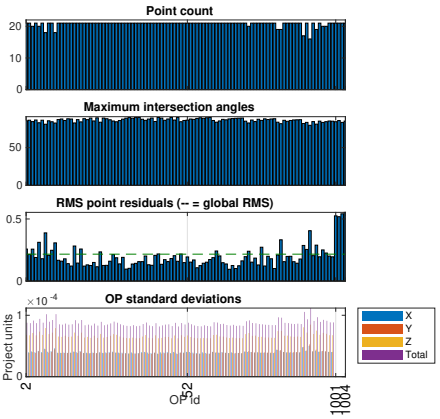

Figure 3: Object point statistics.

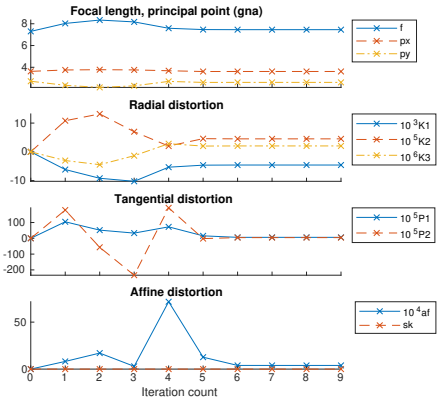

Figure 4: Parameter evolution during the bundle iterations.

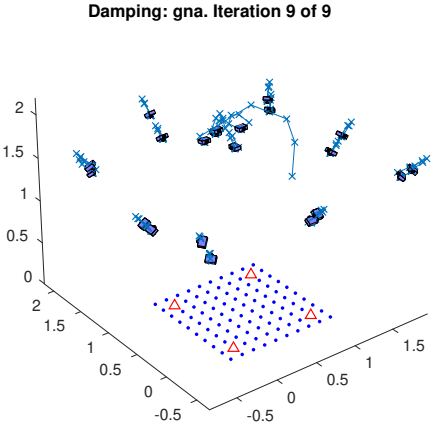

Figure 5: Evolution of the 3D camera network and object points.

```
<meta>
  <name>Camcaldemo</name>
  <date>2019-10-28</date>
  <author>Niclas Börlin</author>
  <project>DBAT</project>
  <purpose>
    Demonstrate camera calibration using the scripting
    feature of DBAT. See also camcaldemo.m in the demo folder.
  </purpose>
  <project_unit>m</project_unit>
  <version>1.1</version>
  <software>
    Software used to generate the data files, e.g., the
    image measurements.
  </software>
  <control_points>
    Information about how the control points were measured.
  </control_points>
</meta>
```

Listing 1: The metadata section is intended for project information but is extendable to contain any structured information.

3. SCRIPT-BASED PROCESSING

Script-based processing uses a single DBAT function `rundbatscript` to load and execute a script. The script is written in the XML language (extensible markup language). Advantages of the XML file format include that they are human-readable, structured, and can easily be extended. Furthermore, since they follow the XML standard, other software may be used for searching or organising the files.

The DBAT XML file contain four major sections: metadata, input, processing, and output.

3.1 Metadata

The metadata section of the XML file is intended for project information, software and hardware that were used to obtain the measurements, the image source, etc. The only DBAT processing of the metadata is to write it in the report file. For an example of a metadata section, see Listing 1.

3.2 Input

The `input` section of a DBAT XML file contain the data and data sources needed for the processing. The main subsections include camera information, control and check information, image information, image measurements and other prior observations.

3.2.1 Camera information The camera information is specified in XML format and can either be present in the main XML file or in a separate file. The basic camera information include the camera name, the camera unit, the sensor and image sizes, and the the nominal focal length. The camera unit specifies the unit used for the internal parameters, typically mm or pixels. For a calibrated camera, the estimated values of the interior orientation parameters are also specified. The aspect parameter can either be computed from the sensor and image sizes or specified directly, in which case the sensor width is computed to match the aspect. The projection model specify what interpretation — Photogrammetry or Computer Vision — of the (Brown, 1971) lens distortion model is used (Börlin et al., 2019), together with the number of radial and tangential coefficients (`nK` and `nP`, respectively). Calibrated distortion coefficients are listed directly (`K` and `P`, respectively). See Listing 2 for some examples.

```
<camera>
  <file>cameras/c4040z.xml</file>
</camera>
<camera>
  <name>Olympus Camedia C4040Z</name>
  <unit>mm</unit>
  <sensor>auto,5.43764</sensor>
  <image>2272,1704</image>
  <aspect>1</aspect>
  <focal>7.5</focal>
  <model>3</model>
  <nK>3</nK>
  <nP>2</nP>
</camera>
<camera>
  <name>Canon EOS 5D</name>
  <unit>mm</unit>
  <sensor>35.96404,24</sensor>
  <image>5616,3744</image>
  <focal>25</focal>
  <cc>24.3581</cc>
  <pp>18.1143, 12</pp>
  <K>2.174e-4, -1.518e-7</K>
  <P>0,0</P>
  <model>3</model>
  <skew>0</skew>
  <aspect>auto</aspect>
</camera>
```

Listing 2: Camera information can be specified either in a separate XML file (top example) or directly in the main XML file (middle example). If the camera is calibrated, the estimated interior parameters are also specified (bottom example).

Blocks in main XML file

```
<ctrl_pts>
  <file format="id,label,x,y,z,sx,sy,sz">sxb-ctrl.txt</file>
  <filter id="351,410">remove</filter>
</ctrl_pts>

<check_pts>
  <file format="id,label,x,y,z,sx,sy,sz">sxb-ctrl.txt</file>
  <filter id="351,410">keep</filter>
</check_pts>
```

sxb-ctrl.txt

```
# Id, Name, X, Y, Z, sigmaX, sigmaY, sigmaZ
317, B2.16,  999604.58, 112344.44, 139.45, 0.02, 0.02, 0.04
351, B4.6 , 1000551.27, 112275.28, 139.86, 0.02, 0.02, 0.04
...
607, B5.21, 1000502.46, 112625.88, 139.64, 0.02, 0.02, 0.04
```

Listing 3: The control and check point blocks typically use a separate comma-separated data source file. The `format` argument allows for flexibility in the format used in the data file. The point sets may be filtered after loading to separate control and check points from the same source file.

3.2.2 Control and check information The control and check information is expected to be listed in "comma-separated" data files, although it is possible to specify other separator characters. A `format` string allows for flexibility in the format used in the data file and is specified as an argument to the file directive (see Listing 3). The typical information to include is the point id, a label, the point coordinates and the coordinate uncertainties. The uncertainties can be specified per coordinate (`sx`, `sy`, `sz`) or jointly (`sxy`, `sxyz`).

The control and check point sections can contain `filter` directives. This enables a single, master, file to be used as a joint control and check point source.

Block in main XML file

```
<images>
  <file format="id,path">images/images.txt</file>
</images>
```

images/images.txt

```
# Image paths for DBAT camera calibration demo
# Format: id, path
 1, data/dbat/images/cam/P8250021.JPG
...
21, data/dbat/images/cam/P8250041.JPG
```

Listing 4: The images subsection refers to a separate text file.

Blocks in main XML file

```
<image_pts>
  <file format="im,id,x,y,sx,sy">mea/imagepts.txt</file>
</image_pts>

<image_pts>
  <file format="im,id,x,y,ignored,ignored" sxy="0.5">
    mea/markpts.txt
  </file>
  <file format="im,id,x,y,ignored,ignored" sxy="1.0">
    mea/smartpts.txt
  </file>
</image_pts>
```

mea/imagepts.txt

```
# Mark points for DBAT camera calibration demo
# Format: image id, point id, x, y, sx, sy
 1,    2, 1429.1, 1456.4, 0.1, 0.1
...
21,   90, 1516.1,   57.9, 0.1, 0.1
```

Listing 5: The image points are read from a comma-separated text data file. In the format string, `im` and `id` refers to the image id and point id, respectively. The uncertainties can be present in the data files (top example) or specified via attributes to the `file` directive. Uncertainty values may be overridden by a combination of the `ignored` format string and uncertainty attributes (bottom example).

3.2.3 Image information The `images` information block uses a similar `file` directive and `format` string as the control information block (see Listing 4). The typical information is the image id and the image path, although only the image id is necessary for the processing. The image path name is only necessary if the images are to be visualised.

3.2.4 Image measurements The image measurements are specified in the `image_pts` subsection and follows the same pattern as the previous (see Listing 5). The typical information in the `format` string is the image id, the point id, the image coordinates, and the measurement uncertainty. Multiple files may be specified as sources by multiple `file` directives. The uncertainty values in the text files may be overridden by a combination of the `ignored` format string and `file` attributes `sx`, `sy`, or `sxy`.

3.2.5 Prior observations Prior EO observations can be specified via the `prior_eo` directive (see Listing 6). Furthermore, if pre-computed initial EO or OP values are to be used by the bundle, the data sources are specified via the `initial_eo` and `initial_op` directives, respectively.

3.3 Operations

The `operations` section contain information about the processing to be applied to the input data. The operations are executed in the specified order. A typical operation sequence is:

```
<prior_eo>
  <file format="id,x,y,z,omega,phi,kappa,sx,sy,sz,so,sp,sk"
        angle_units="deg">
    prior/prior_eo.txt
  </file>
</prior_eo>

<initial_eo>
  <file format="id,x,y,z,omega,phi,kappa" angle_units="deg">
    prior/initial_eo.txt
  </file>
</initial_eo>

<initial_op>
  <file format="id,x,y,z">prior/initial_op.txt</file>
</initial_eo>
```

Listing 6: Prior observations and pre-computed initial values can be similarly specified.

1. Apply a sanity check on the input.
   (a) Optionally, filter points on low ray count.
2. Set up initial values for the bundle.
   (a) Optionally, apply secondary sanity checks and/or filtering given the initial values.
3. Specify what parameters the bundle should estimate.
4. Execute the bundle.

3.3.1 Sanity checks The sanity checks can be used to catch blunders in the data files, poor initial values, and network problems. The supported sanity check operations are

`check_ray_count` Check that no point has too few rays. Can be used to catch, e.g., missing points in the data files.

`check_ray_angles` Check that no point has a too small ray intersection angle. Requires initial or estimated values for the camera and object point positions. Can be used to understand why a bundle fails.

`check_projection` Check whether the object points are projected outside the images. If used on initial values and a wide tolerance, the operation can catch grossly incorrect initial values. If used on the final values and a strict tolerance, the operation can detect object points that have converged to an incorrect position behind a camera.

`check_structural_rank` Check whether the structural rank of the Jacobian (design matrix) is too low. Can be used to, e.g., detect holes in the network.

`check_numerical_rank` Check whether the numerical rank of the Jacobian (design matrix) is too low. Can be used as a debugging tool to understand why a bundle operation failed.

3.3.2 Filtering The `filter_points` operation can be applied to remove points that have too few rays and/or a too small intersection angle. The latter operation requires that the EO and OP positions have been estimated.

3.3.3 Set initial values The initial values can be set by specifying constant or pre-loaded values or as a result of performing a photogrammetric computation. The `set_initial_values` directive allows the user to set individual or groups of parameters to specific values or to use pre-loaded values. An alternative is to use the `spatial_resection` and/or `forward_intersection` operations. Spatial resection requires initial values of the IO parameters and measurements of control points. Forward intersection requires initial values of the IO and EO parameters.

3.3.4 Specify parameters to estimate The parameters to estimate are set by the `set_bundle_estimate_params` directive. It allows the user to specify exactly parameters should be estimated by the bundle and what parameters should be treated as fixed. A standard bundle would estimate the EO and OP parameters only. A self-calibration project would also estimate some or all IO parameters.

As a special case, if the datum is not set by the control points, the `set_datum` operation can set the datum. Currently, only the dependent relative orientation is supported.

3.3.5 Execute the bundle The `bundle_adjustment` operation executes the bundle adjustment. This is typically the last operation specified.

3.3.6 A full operations example Listing 7 shows the full operational flow for a camera calibration script. Since the camera calibration situation is assumed to be reasonably controlled, only a single sanity test is included to catch major blunders, such as to include the wrong files. For less controlled settings, more sanity checks and/or filtering might be necessary.

```
<operations>
  <operation min_rays="2">check_ray_count</operation>
  <set_initial_values>
    <io>
      <cc>focal</cc>
      <others>default</others>
    </io>
  </set_initial_values>
  <operation>spatial_resection</operation>
  <operation>forward_intersection</operation>
  <set_bundle_estimate_params>
    <io>
      <all>true</all>
      <skew>false</skew>
    </io>
    <eo>
      <all>true</all>
    </eo>
    <op>
      <all>true</all>
    </op>
  </set_bundle_estimate_params>
  <operation>bundle_adjustment</operation>
</operations>
```

Listing 7: The full operation sequence for a camera calibration. After an initial ray count sanity check, the camera constant is initialised to the nominal focal length. The other parameters are set to default values (principal point at the image center, unit aspect, zero skew and zero lens distortion). The initial EO parameters are computed by spatial resection, followed by forward intersection to set the initial object point coordinates. All OP, EO, and IO parameters except skew are estimated by the bundle.

```
<output>
  <plots>
    <plot id="1">image</plot>
    <plot>image_stats</plot>
    <plot max_op="1000">op_stats</plot>
    <plot convex_hull="true">coverage</plot>
    <plot>params</plot>
    <plot cam_size="0.1">iteration_trace</plot>
  </plots>
  <files base_dir="$HERE">
    <report_file>
      <file>result/report.txt</file>
    </report_file>
    <io>
      <file>result/c4040z.xml</file>
    </io>
    <image_residuals top_count="50">
      <file>result/top_residuals.txt</file>
    </image_residuals>
  </files>
</output>
```

Listing 8: The `output` section specifies plots and result files of interest for the user. This example contains image point observations overlaid on the image (see Figure 1), image and object point statistics (figures2–3), the image coverage, the evolution of the camera parameters during the bundle (Figure 4), and a 3D evolution of the camera network and the object points (Figure 5). The result files include a standard summary report file, the calibrated camera, and a text file that contain the top 50 largest image residuals.

3.4 Output

The `output` section contain information about what kind of results are of interest for the user. This includes different kinds of plots and files. The plotting can either show some quality parameters graphically, e.g., image coverage, or show how the parameter estimates have evolved during the bundle iterations, see section 2.2.5. Output files include report files that summarise the processing, DBAT XML camera files to store calibrated cameras, or comma-separated table files that can be imported into other software.

3.5 Data integrity

During processing of all input text data files, each non-blank, non-comment line in the data file is expected to match the `format` string. To reduce the probability of accidentally processing bad data, any mismatch between the number of expected and actual fields triggers an error.

3.6 File structure

The `input` and `output` sections may have an attribute `base_dir` to specify a common base directory for the files. Non-absolute path names within the same section are assumed to be relative to the specified base directory. This simplifies grouping of data files while retaining flexibility for special cases.

3.7 Language extensions

The combination of the XML language and the Matlab language makes it easy to extend the DBAT scripting language. An extension with a new operation requires adding two things: A Matlab function that performs the operation and adding the name of the operation to the list of known operations of the

script parser. Any parameters to the function can be specified as XML attributes to the operation. Adding a parameter to an existing function would only require the latter. Similarly, extending the input capabilities may simply require adding a keyword to the format string and support for the keyword in the corresponding Matlab function.

4. CONCLUSIONS AND FUTURE WORK

4.1 Conclusions

The modularity provided by the scripting allows a simple setup of the DBAT processing with little to no programming knowledge. At the same time, the language provides flexibility of what parameters to estimate or what operations to perform and in what order. Furthermore, the ability to read text files with a general format enables input from many sources. The combination of the XML and Matlab languages makes it relatively easy to add new function or file format to the DBAT scripting language. These qualities suggest that DBAT XML scripting should be useful both for users that want to independently verify a black-box computation by another software, or teach students the inner workings of the bundle adjustment process. The modularity concept is particularly interesting for teaching purposes, as it enables a step-by-step learning of the photogrammetric workflow in lieu of a one-click black-box solution. The implementation of the XML file also presents a simple interface with which the workflow may be explained systematically.

4.2 Future work

Future work include adding scripting support for multiple cameras per project, something already supported by the DBAT back-end. Additional future work include adding support for binary input and output formats and the input and output of full covariance matrices.

REFERENCES

Börlin, N., Grussenmeyer, P., 2013a. Bundle Adjustment with and without Damping. *Photogrammetric Record*, 28(144), 396-415.

Börlin, N., Grussenmeyer, P., 2013b. Experiments with Metadata-derived Initial Values and Linesearch Bundle Adjustment in Architectural Photogrammetry. *ISPRS Annals of the Photogrammetry, Remote Sensing, and Spatial Information Sciences*, II-5/W1, 43-48.

Börlin, N., Grussenmeyer, P., 2014. Camera Calibration using the Damped Bundle Adjustment Toolbox. *ISPRS Annals of the Photogrammetry, Remote Sensing, and Spatial Information Sciences*, II(5), 89-96. Best paper award.

Börlin, N., Grussenmeyer, P., 2016. External Verification of the Bundle Adjustment in Photogrammetric Software using the Damped Bundle Adjustment Toolbox. *International Archives of Photogrammetry, Remote Sensing, and Spatial Information Sciences*, XLI-B5, 7-14.

Börlin, N., Murtiyoso, A., Grussenmeyer, P., Menna, F., Nocerino, E., 2018. Modular Bundle Adjustment for Photogrammeric Computations. *International Archives of Photogrammetry, Remote Sensing, and Spatial Information Sciences*, XLII(2), 133-140.

Börlin, N., Murtiyoso, A., Grussenmeyer, P., Menna, F., Nocerino, E., 2019. Flexible Photogrammetric Computations using Modular Bundle Adjustment. *Photogrammetric Engineering and Remote Sensing*, 85(5), 361-368.

Brown, D. C., 1971. Close-range camera calibration. *Photogrammetric Engineering*, 37(8), 855–866.

Cramer, M., 2006. The ADS40 Vaihingen/Enz geometric performance test. *ISPRS Journal of Photogrammetry and Remote Sensing*, 60, 363-374.

Lourakis, M. I. A., Argyros, A. A., 2009. SBA: A Software Package for Generic Sparse Bundle Adjustment. *ACM Transactions on Mathematical Software*, 36(1), 30 pp.

Lumban-Gaol, Y. A., Murtiyoso, A., Nugroho, B. H., 2018. Investigations on the bundle adjustment results from SfM-based software for mapping purposes. *International Archives of Photogrammetry, Remote Sensing, and Spatial Information Sciences*, XLII(2), 623.

Menna, F., Nocerino, E., Drap, P., Remondino, F., Murtiyoso, A., Grussenmeyer, P., Börlin, N., 2018. Improving Underwater Accuracy by Empirical Weighting of Image Observations. *International Archives of Photogrammetry, Remote Sensing, and Spatial Information Sciences*, XLII(2), 699-705.

Murtiyoso, A., Grussenmeyer, P., Börlin, N., 2017. Reprocessing Close Range Terrestrial and UAV Photogrammetric Projects with the DBAT Toolbox for Independent Verification and Quality Control. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-2/W8, 171–177.

Murtiyoso, A., Grussenmeyer, P., Börlin, N., Vandermeerschen, J., Freville, T., 2018. Open Source and Independent Methods for Bundle Adjustment Assessment in Close-Range UAV Photogrammetry. *Drones*, 2(1). http://www.mdpi.com/2504-446X/2/1/3.

Rupnik, E., Daakir, M., Pierrot Deseilligny, M., 2017. MicMac – a free, open-source solution for photogrammetry. *Open Geospatial Data, Software and Standards*, 2(1), 14.

Snavely, N., Seitz, S. M., Szeliski, R., 2008. Modeling the World from Internet Photo Collections. *International Journal of Computer Vision*, 80(2), 189-210.

Wu, C., Agarwal, S., Curless, B., Seitz, S. M., 2011. Multicore bundle adjustment. *Proceedings of the CVPR*, 3057–3064.

APPENDIX

This appendix presents a full example script for camera calibration. The example includes the main XML file and the associated text files.

Main XML file

```
<?xml version="1.0" encoding="UTF-8"?>
<document dbat_script_version="1.0">
  <meta>
    <name>Camcaldemo</name>
    <date>2019-10-23</date>
    <author>Niclas Börlin</author>
    <version>1.0</version>
    <project>DBAT</project>
    <purpose>
      Demonstrate camera calibration using the scripting
      feature of DBAT. See also camcaldemo.m in the demo
      folder.
    </purpose>
    <software>
      Measurements: Photomodeler scanner 2015.
    </software>
    <control_points>
      Synthetic control points.
    </control_points>
  </meta>

  <input base_dir="$HERE">
    <ctrl_pts>
      <file format="id,label,x,y,z,sxyz">
        reference/camcal-ctrl.txt
      </file>
    </ctrl_pts>

    <images>
      <file format="id,path">
        images/images.txt
      </file>
    </images>

    <image_pts>
      <file format="im,id,x,y,sxy">
        measurements/markpts.txt
      </file>
    </image_pts>

    <cameras>
      <camera>
        <name>Olympus Camedia C4040Z</name>
        <unit>mm</unit>
        <sensor>auto,5.43764</sensor>
        <image>2272,1704</image>
        <aspect>1</aspect>
        <focal>7.5</focal>
        <model>3</model>
        <nK>3</nK>
        <nP>2</nP>
      </camera>
    </cameras>
  </input>

  <operations>
    <operation min_rays="2">check_ray_count</operation>

    <set_initial_values>
      <io>
        <cc>focal</cc>
        <others>default</others>
      </io>
    </set_initial_values>

    <operation>spatial_resection</operation>
    <operation>forward_intersection</operation>

    <set_bundle_estimate_params>
      <io>
        <all>true</all>
        <skew>false</skew>
      </io>
      <eo>
        <all>true</all>
      </eo>
      <op>
        <all>true</all>
      </op>
    </set_bundle_estimate_params>

    <operation>bundle_adjustment</operation>
  </operations>

  <output>
    <plots>
      <plot id="1">image</plot>
      <plot>image_stats</plot>
      <plot max_op="1000">op_stats</plot>
      <plot convex_hull="true">coverage</plot>
      <plot>params</plot>
      <plot cam_size="0.1">iteration_trace</plot>
    </plots>
    <files base_dir="$HERE">
      <report_file>
        <file>result/report.txt</file>
      </report_file>
      <io>
        <file>result/c4040z.xml</file>
      </io>
      <image_residuals top_count="50">
        <file>result/top_residuals.txt</file>
      </image_residuals>
    </files>
  </output>
</document>
```

camcal-ctrl.txt

```
# Control point coordinates for Photomodeler camera
# calibration sheet.
# Id,label,X,Y,Z,sxyz
1001, CP1, 0, 1, 0, 0.01
1002, CP2, 1, 1, 0, 0.01
1003, CP3, 0, 0, 0, 0.01
1004, CP4, 1, 0, 0, 0.01
```

images.txt

```
# Image paths for DBAT camera calibration demo
# Format: id, path
 1, data/dbat/images/cam/P8250021.JPG
 2, data/dbat/images/cam/P8250022.JPG
...
20, data/dbat/images/cam/P8250040.JPG
21, data/dbat/images/cam/P8250041.JPG
```

markpts.txt

```
# Mark points for DBAT camera calibration demo
# Format: image id, point id, x, y, sxy
 1,    2, 1429.1, 1456.4, 0.1
 1,    3, 1217.8, 1456.1, 0.1
...
21,   92, 1358.1,   61.8, 0.1
21,   90, 1516.1,   57.9, 0.1
```