NASA/CR–2014-218518

Photogrammetry Toolbox Reference Manual

Tianshu Liu
Western Michigan University, Kalamazoo, Michigan

Alpheus W. Burner
Jacobs Technology Inc., Hampton, Virginia
Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA’s STI. The NASA STI program provides access to the NASA Aeronautics and Space Database and its public interface, the NASA Technical Report Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- **TECHNICAL PUBLICATION.** Reports of completed research or a major significant phase of research that present the results of NASA Programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers, but having less stringent limitations on manuscript length and extent of graphic presentations.

- **TECHNICAL MEMORANDUM.** Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- **CONTRACTOR REPORT.** Scientific and technical findings by NASA-sponsored contractors and grantees.

- **CONFERENCE PUBLICATION.** Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- **SPECIAL PUBLICATION.** Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- **TECHNICAL TRANSLATION.** English-language translations of foreign scientific and technical material pertinent to NASA’s mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at [http://www.sti.nasa.gov](http://www.sti.nasa.gov)

- E-mail your question to help@sti.nasa.gov

- Fax your question to the NASA STI Information Desk at 443-757-5803

- Phone the NASA STI Information Desk at 443-757-5802

- Write to:
  STI Information Desk
  NASA Center for AeroSpace Information
  7115 Standard Drive
  Hanover, MD 21076-1320
Photogrammetry Toolbox Reference Manual

Tianshu Liu  
*Western Michigan University, Kalamazoo, Michigan*

Alpheus W. Burner  
*Jacobs Technology Inc., Hampton, Virginia*

September 2014
The use of trademarks or names of manufacturers in this report is for accurate reporting and does not constitute an official endorsement, either expressed or implied, of such products or manufacturers by the National Aeronautics and Space Administration.
Abstract

Specialized photogrammetric and image processing MATLAB functions useful for wind tunnel and other ground-based testing of aerospace structures are described. These functions include single view and multi-view photogrammetric solutions, basic image processing to determine image coordinates, 2D and 3D coordinate transformations and least squares solutions, spatial and radiometric camera calibration, epipolar relations, and various supporting utility functions.

Introduction

Photogrammetric techniques have been found to be very useful for specialized measurements of component deformation of advanced aircraft during ground or in-flight testing as well as deformation of large space structures during ground testing. The basis of photogrammetry can be summarized as the determination of a parameter or parameters of interest in 3D object space from 2D image coordinates. These parameters could be spatial coordinates (1D to 3D), deformation, angle, or changes in angle, etc. With the recent replacement of film with electronic image sensors, some authors have used expressions other than photogrammetry to denote this extraction of spatial information from images. Part of the impetus for this name-change is to emphasize the modern nature of these efforts and to emphasize that digital images, rather than film, make up the raw data. These various names, which are largely a matter of personal choice of the authors of a given publication, include digital photogrammetry, geomatics, videogrammetry, videometrics, and computer vision. It remains to be seen which term will eventually be considered the defining one if the long-standing term photogrammetry is indeed supplanted by another more meaningful term.

Classic photogrammetry previously consisted of photographs that were read on a monocomparator in order to extract image coordinates. A computer was then used for data reduction. Currently electronic images are acquired and reduced with automated image processing, often on the same computer and often with many images in a time sequence or set of time sequences. Some of the specialized aerospace applications that the development of the photogrammetry toolbox is directed towards include aeroelastic model deformation, wind tunnel model attitude, sting bending, the study of model injection rates at blow-down facilities, determination of model position, deformation of micro air vehicles, deformation of aircraft in-flight, structural deformation of ultralight and inflatable large space structures, etc. In addition a whole class of advanced imaging flow diagnostic and visualization techniques either use some form of photogrammetry or could benefit from its use. These image-based flow diagnostic techniques include pressure and temperature sensitive paints (PSP/TSP), Doppler global velocimetry (DGV), particle image velocimetry (PIV), projection moiré interferometry (PMI), planar laser induced fluorescence (PLIF), and laser-induced thermal acoustics (LITA).

The Photogrammetry Toolbox (PT) described here should be viewed as complementing rather than replacing standard photogrammetric packages that are used quite commonly for spatial measurements where the images can be acquired in a sequential manner as the camera is moved about the object. Instead the PT was developed for specialized aerospace applications where traditional photogrammetry techniques are often not applicable due to various constraints such as limitations on camera location, limitations on size and mass of the camera, requirement for remote operation, severe limits on setup time due to wind tunnel productivity requirements, and/or the need for near real-time results. The functions in the PT serve as building blocks to
develop custom measurement systems that may utilize non-traditional photogrammetry for near real-time applications. The functions can be relatively easily customized to further enhance their value in the development of measurement systems for unique and varied applications. In some cases the functions can be utilized within the MATLAB environment for the application. In other cases where performance is critical, the functions can be used to develop the measurement strategy, which can then be implemented in C-code to maximize efficiency. Although the MATLAB Image Acquisition toolbox was not utilized in the current version of the PT, it is anticipated that the coupling of the PT with the acquisition toolbox should provide a powerful developmental platform.

**Toolbox Folders**

The primary folder containing the PT functions is entitled *Photogrammetry Toolbox*. There are three subfolders located within the primary folder. The first subfolder *Documentation* contains document pages for each function written in Microsoft Word. The document pages for each function cover the purpose, syntax, arguments, output, additional remarks, example scripts, and equations. The second subfolder *Example Scripts* contains scripts that can be run from the MATLAB Command Window to illustrate the usage of the various functions. The naming convention for the example scripts is the function name with *Example* appended to the end of function name. For instance, the example script for the function *resection* is named *resectionExample*. (The files containing the functions and scripts have a .m extension which should be assumed in any file names for functions or scripts within this document.) The third subfolder *Sample Files* contains data and digital image files that are utilized within the example scripts. It is recommended that the primary folder and its three subfolders be placed in a convenient location within *My Documents* to facilitate file backup and to more easily incorporate the PT functions when upgrading to a newer version of MATLAB. The *Photogrammetry Toolbox* folder and its subfolders should be added to the top of the MATLAB path. By adding to the top of the path, m-files in the PT will override any conflicting names lower in the path. However, it is still recommended that conflicting function names be eliminated to avoid confusion. The folders can be added to the path from within MATLAB by selecting *Set Path*... under *File*, select *Add Folder*..., select *Add with Subfolders*..., and then select *Save*. Typing *path* at the Command Prompt should show the *Photogrammetry Toolbox* and its subfolders at the top of the path. Once the primary and three subfolders are added to the path, the functions and example scripts can be invoked from any folder (except for the special example script *camcal_goldenExample* which can only be run from the primary PT folder). A folder contents feature available in MATLAB enables all the function names in a folder to be listed in the Command Window with one row per m-file. Each row contains the name of the function or script and a brief 1-line description of the purpose of the file. The name of the m-file is an active link to quickly obtain the more detailed multi-line help information normally provided at the top commented segment of the m-file. A short script entitled *helpPT* can be invoked from any folder to quickly and conveniently review the m-files that are available in the Photogrammetry Toolbox folder and to access more detailed information by selecting any function in the list. The input to the *helpPT* function comes from the special script *contents* consisting of all comments located in the primary PT folder. The script *contents* is created (or edited) by running *Contents Report* from the *Current Directory Browser*.

**Overview of Toolbox Functions**

The toolbox contains functions for elementary analysis of digital images to determine image plane coordinates, camera calibration suitable for aerospace applications, single-view and multi-view determination of object space coordinates, determination of camera pointing angles and location, 2D and 3D coordinate
transformations along with functions to determine transformation coefficients given 2 sets of object space
coordinates, and assorted utility functions. The functions were developed in MATLAB version 2006a, but
should be applicable for some older versions as well. Most of the image processing functions make use of the
Image Processing Toolbox, which must be present to utilize those functions. Following this overview, each
function, listed in alphabetical order, is described in more detail within its own document page. The
document pages for the functions cover the purpose, syntax, arguments, output, additional remarks, example
scripts, and equations. To ensure continuity from previous work at NASA Langley, some of the functions
utilize the familiar camera input file consisting of a column of input calibration coefficients that must be
entered in a prescribed order. Other functions utilize a structure for input which has the advantage of
automatically documenting any MATLAB scripts that call the functions. Another advantage of the structure
for input arguments is that the order of the entry is irrelevant since the field labels of the structure dictate
which coefficients are intended to be passed to the function. The use of structures also improves the
conciseness of the calling syntax and tolerates more fields than needed for the function arguments (for
instance for documentation of the experiment in a notes field which will be ignored by the function that is
invoked). This use of structures in the calling syntax of the functions is expected to aid in the usability and
applicability in future developments. The functions loadCamStruct and saveCamStruct are utilities to load
and save camera parameter structures in text format for use outside MATLAB.

The simulation functions collinearity and XYZ2xy are used to create ideal image plane data corresponding to
a set of object coordinates given various camera parameters. The function collinearity uses structures for
input and output (typically in mm) whereas XYZ2xy uses a column entry for the camera parameters, including
distortion with output in pixels. The function distortApply can be used to apply distortion to the output from
collinearity. The function mm2pixel can be used to convert the output of collinearity from mm to pixels. A
complementary function pixel2mm is used to convert from pixels to mm. The function xyplot can be used to
compare calculated and measured image plane coordinates.

Digital image analysis functions include several simple, but useful image processing functions that enable
manual selection of targets or locations on a digital image (pixelXYselect) and enable one to establish the
maximum gray scale on the perimeter of a rectangular region of interest for use in background removal
(findBackground). Other image processing functions enable the computation of gray scale centroids
(centroid, centroid_cal_fun, clicking_targ_fun, location_target1_fun) and display of gray scale to the
Command Window (displayGrayScale), given image locations, regions of interest, and possible background
for removal before centroiding or display. The function GrayScaleDisplay displays the image in the top half
of a figure window along with an interactive pixel grayscale display in the lower half. The function has a
single input argument that can be either an image variable currently in the workspace or a character string or
variable that represents a valid image file name. The function opens an image file dialog box for file selection
if invoked without an input argument for convenient examination of digital images. Pixel location and
grayscale are displayed in the figure window as the cursor is moved over the image itself or over the display
of grayscale. A small rectangular box overlay on the image, which indicates the coverage of the grayscale
display area, can be moved about the image to examine in detail the grayscale of any portion of the image.
This function is very convenient and easy to use for examining grayscale of any image and complements the
function that displays grayscale to the Command Window (displayGrayScale). The function roiSelect
enables a single or multiple regions of interest (roi) of an image to be selected by mouse. The single input
argument can be an image variable or file name. The function opens an image file dialog box for file
selection if invoked without an input argument. The rectangular roi is selected by positioning the cursor to
one corner of the desired rectangular area, pressing the left mouse button, and dragging to the other corner of
the rectangle. A single roi or many roi’s can be selected. Optionally the function can output an image which
has the original grayscale of the input image, but with the grayscale in each roi set to zero. In this case the
output image would consist of rectangular patches of black on the grayscale of the original input image. The
newly created output image is displayed in its own figure window. The function should be useful for cases in
which the targets of interest are in a limited area of a cluttered image. A similar function uses a single polygon roi instead of rectangles (roiPolySelect). This function allows for selection of odd-shaped regions that might be awkward to select with several rectangular roi’s. The polygon roi function also returns an image that is the same size and class as the input image, but with only the polygon roi containing grayscale from the original input image. The rest of the output image outside of the polygon roi is set to zero. The inverse image is also available in which the polygon roi is black (grayscale = 0), but the rest of the input image is intact. Both of these functions should be useful to eliminate troublesome areas of an image before further processing in cases where automated image processing over the whole image fails. Several epipolar functions (epipolarLine_x, epipolarLine_y, epipolarRelation_x, epipolarRelation_y) enable the matching of a target from one image with the corresponding target from a second image, which can help with automated analysis of 2-view photogrammetric image data.

The GUI function imagePrelim serves as a preliminary tool for automated target location on digital images. The GUI utilizes the regionprops (IPT) function from the Image Processing Toolbox that operates on binary images. (Functions from the Image Processing Toolbox are followed by IPT enclosed in parentheses.) A pushbutton enables selection of the appropriate digital image file (via a popup file selection window) for loading and displaying in a figure window within the GUI. The image is displayed in grayscale, but all preliminary processing is accomplished with a binary version of the image. The initial threshold for the binarization when the image file is first imported is determined by the graythresh (IPT) function. A label image is then created from the binary image using bwlabel (IPT). The regionprops (IPT) function is then used to create a structure containing the binary centroids and bounding boxes of each labeled region within the label image. The bounding boxes for each potential target (some of which may potentially be false targets) are overlaid on the image. A larger cross is plotted for very small (and usually false) targets smaller than 3 pixels to improve their identification. The number of targets found, as well as the relative threshold (ranging from 0 to 1), are displayed. A slider box (with display) can then be used to interactively adjust the threshold. The newly found targets based on the just selected threshold are overlaid on the image so that one can interactively quickly determine a suitable threshold to automatically find all the valid targets. Typically the highest threshold that finds all the valid targets is selected before possible further processing with the GUI (if additional false targets are found). Slider bars for minimum and maximum bounding box size can then be used to interactively limit the targets found. Selection of a new image or threshold for the current image reinitializes the process. A pushbutton can be used to invert the grayscale before inputting a digital image file for cases with black targets on a white background. The file name of the inputted digital image is displayed on the GUI along with the number of targets found. Another pushbutton initiates the selection of a polygon region of the image (using roiPolySelect) in order to remove regions of the image that might contain false targets that are especially hard to discriminate with threshold or size limits. Target ID numbers can be overlaid on the image using overlayCentroidsBox and the preliminary binary centroid data can be saved in text format (with user selected file name via file dialog box) with point number, x and y centroid data, half-width, and half-height of each bounding box respectively. This capability is useful in addition when the binary file is used as input (for start values) for full grayscale centroiding. A toggle button can be used to show the binary image without processing to aid in preliminary analysis of cluttered images since the processing can be very time consuming when using regionprops (IPT) at each change of the grayscale threshold. Thus an appropriate threshold can be determined by examination of the binary image before initiating the processing via the regionprops (IPT) function. In this mode all processing except for the slider threshold is disabled until the get image file pushbutton is activated to restart the process. An additional pushbutton allows for manual selection (via mouse) of target ID numbers and the subsequent saving of that xpixel and ypixel data along with the corresponding target ID as a text file (with user selected file name via file dialog box). This additional pushbutton should help in cases where the automatically generated centroid data does not have the desired numbering system. A button panel allows the selection of a centroid file to be overlaid on the image. For the overlay it is assumed that the first three columns of the data from the file are in order target ID, x, and y. The next 2 columns, if they exist, are taken to be the half-height and half-width of
the bounding boxes. A text entry box is available to specify a single value for the bounding box width and height for files of only 3 columns, which is then used in the overlay plot for all targets. Both the bounding boxes and target IDs are plotted in a color chosen from a popup menu of color selections to aid in discrimination of multiple plots overlaid on the same image. Another button panel allows 2 centroid files to be combined into a new file, getting the correct target IDs from 1 file and the correct centroid data from another. The match tolerance (x, y pixel values must be within this set tolerance to match) is set from within an edit box. Another button panel allows grayscale centroiding (with automated background removal based on the max grayscale on the perimeter of the bounding box) and output to a new file. This panel is convenient for computing grayscale centroids using the binary centroid files created within the GUI itself as start values. An additional width and height to be added to the binary bounding boxes is entered through an edit box. This helps to minimize clipping of the target since grayscale below the threshold (set to zero during the binarization of the image) may be outside the bounding box found from the binary image, but still may be a valid part of the target. Another button panel gives the option of taking threshold and size restrictions from the edit boxes corresponding to the sliders. A separate process button within the panel must be pressed to initiate image processing based on the values in the edit boxes. (The sliders for threshold, min size, and max size are ignored if the edit boxes radio button is selected. When the process button is selected, the values for threshold, min size, and max size are then taken from the corresponding edits boxes as entered by the user instead of from the sliders.) This greatly speeds up preliminary investigations with large format images of several megapixels compared to slider selection. (Since with the sliders activated computations are made at intermediate positions as the sliders are moved toward their final destinations.)

The camera calibration functions include several utilizing optimization of a single view of a 3D calibration block to provide a very useful simplified method to determine the major camera parameters necessary for photogrammetric measurements. The optimization functions include camcal_fun and camcal_fun_1 and support functions dlt, dlt0, lleast, lleast3, residual_exterior, residual_interior1, residual_interior2, resec, resec3, and resecA. The script camcal_goldenExample utilizes the convenience of the MATLAB environment for input and output while invoking three executables for camera calibration by optimization using the Golden search method. The script will normally only run properly from the primary PT folder. The files calibrator.exe, plot.exe, and simulator.exe must be copied from the primary PT folder to another folder for proper operation in other than the primary PT folder. Note that single view camera calibration is relatively quick and convenient, but may not be the best camera calibration available. If the ultimate in camera calibration is required one of the commercially available photogrammetric packages should be considered. Note that for some specialized aerospace applications, such as the single view determination of model deformation, camera calibration is not the primary calibration, but rather a preliminary or partial calibration to reduce nonlinearities of the final calibration. Thus camera calibration in those cases is not as critical as for traditional photogrammetry. For instance, the final calibration for single view model deformation consists of an angle calibration based on an onboard inertial device. The camera calibration primarily reduces the nonlinearities and the amount of correction that the final angle calibration must accommodate. The use of the quick single view camera calibration by optimization reduces setup time and increases wind tunnel productivity compared to traditional photogrammetric camera calibration. Additional camera calibration functions enable the determination of the camera constant (cameraConstant) and distortion coefficients (distortSolve). Once the the radial and decentering distortion terms are found, the function distortCorrect can be used to correct image coordinates in mm. The function to determine the camera constant typically requires 2 or more images of a calibration fixture (which can be planar and is approximately perpendicular to the optical axis of the camera) at known displacements from the camera. The values of the photogrammetric principal point and distortion coefficients must be known (or entered as zero for initial results). Advantages of this function is that an estimate of precision is computed for the camera constant from the least squares and projective coupling between other camera parameters is lessened (see Appendix). The function to solve for the distortion coefficients can be applied to a single image of a planar target fixture. Precision estimates of the coefficients from least squares method are also computed. These 2 functions should serve as useful
complements to the optimization functions, depending on the application. Radiometric camera calibration is possible with the two functions RadiomCali_cheby_fun and RadiomCali-poly_fun. These two functions determine the camera response function given two images taken at different f-numbers so that nonlinearity in the grayscale versus irradiance can be greatly corrected for situations where a linear response is critical.

Functions were developed to allow for single-view (singleView, xy2XZ) and multi-view (intersection, xy2XYZ) determination of object coordinates. The function singleView enables single view solutions for one coordinate (X, Y, or Z) or pairs of coordinate (X-Y, X-Z, Y-Z). The function can handle cases in which either the image or object coordinate data has target point numbers not found in both, with only the valid solutions for target numbers common to both image and object outputted. A structure format is used for the object coordinate data in which the field representing the particular coordinate(s) to be solved are entered as null array(s) such as XYZ.X = []. The output of the new function is an N x 4 array when 2 coordinates are solved for (which are found from 2 equations in 2 unknowns) containing N target point numbers and X, Y, and Z coordinates (echoing the input known coordinate in the output array). The output for single coordinate solutions is an N x 5 array, where the 5th column is the estimated standard deviation computed from the least squares solution of 2 equations in 1 unknown. The redundancy of this solution is weak with only 1 degree of freedom, but the computed standard deviations can be useful for comparisons and are useful in a global sense by examining the mean value of the standard deviation for a given data set. Limited numerical tests indicate that this least squares estimate of the standard deviation of the single coordinate solutions reasonably represents the object coordinate random error due to image plane error (although it typically underestimates the error by about 25% on average), but grossly overestimates the error due to random errors in the input object coordinates. The intersection function determines 3D coordinates given image plane coordinates and camera parameters from two or more views. A structure array is used for input that allows for compact and flexible input of camera parameters and image coordinates from multiple cameras or views. The function, which can handle any number of cameras or views, accommodates for missing or extra image coordinates. The output of the function is an 8 column numeric array that has the number of rows corresponding to the number of image points that are seen by at least two views. The first column contains the image target point number, columns 2 to 4 contain in order X, Y, and Z, columns 5 to 7 contain in order Xc, Yc, and Zc, which are the estimates of the standard deviation of the spatial coordinates from the least squares reduction, and column 8 contains the number of views used in the reduction for each point. As for the single coordinate computations within singleView, the redundancy is weak (only 1 degree of freedom) for the estimates of the standard deviations. However, these estimates are still useful, both in a global sense by examining their mean values, and for identifying possible outliers.

The function resection uses nonlinear least squares to determine camera pointing angles and location. The camera parameters $\omega, \phi, \kappa, X_c, Y_c, Z_c$ are found, given image coordinates, object coordinates, and camera interior parameters $(c, x_0, y_0)$. Since the camera constant is not treated as an unknown, the resection function works on planar target fields, which can be very useful in aerospace applications. Common target point numbers are found for the image and object coordinates for the solution, allowing for the image or object coordinates to be a subset of either. Estimates of the standard deviation of the parameters are returned from the function, along with the global standard deviation of unit weight. Also returned are the standard deviations of the differences in the $x$- and $y$-image coordinates comparing the input image with the coordinates computed from the input object and outputted resection parameters using collinearity. Thus a set of coefficients are passed back to the calling script to help in assessing the quality of the results. The use of a structure for output allows for echoing of input data that is not solved for, along with the solved for values and supporting statistics. Since the fields of the output structure include those used in other functions (such as intersection) the output of the resection function can then be passed directly as input to other functions in the toolbox for further computations. Any extra fields not required by a particular toolbox function are simply ignored by that function. Another function (resecc_ZW) provides a closed-form solution for resection that
does not need initial guesses (developed by Zeng and Wang in 1992). The function needs only object and image data from 3 target points that are not collinear to determine exterior orientation parameters $o$, $\phi$, $\kappa$, $X_s$, $Y_s$, and $Z_s$. The function, which returns 2 possible sets of exterior parameters, should be called twice to isolate the correct solution, so that in practice 4-target points are actually required. In the second call to the function, one of the target points is replaced. The correct solution is then found as the solution common to both sets within some tolerance. This function should be useful to complement nonlinear least squares resection functions as well as camera calibration by optimization.

Several functions are included for application and solving of 2D and 3D coordinate transformations. Included are forward and inverse conformal (conformal2D, conformal2Dinv, conformal3D, conformal3Dinv) and 2D affine transformations (affine2D) as well as linear and nonlinear least squares functions to find the 2D and 3D conformal (conformal2DLLS, conformal2NLLS, conformal3DNLLS) and 2D affine parameters (affine2DLLS), along with estimates of their standard deviation, given two sets of coordinates. All functions utilize target point numbers in column 1 for the input data sets to enable the selection of common target point numbers for computation. Thus each data set can have either missing or extra target point numbers without negatively impacting the solution. Nonlinear least squares (NLLS) functions are included for the 2D and 3D conformal transformations which are able to selectively solve for or treat as constant any or all of the unknown parameters. Tolerances can also be placed on any of the unknowns to restrict the range of variation within the NLLS computations. The tolerancing should be used with care since its implementation within the function may not yield correct estimates of the standard deviations of the various parameters. It is recommended that if tolerancing leads to a parameter being driven to 1 edge of the hard-clip limits (and that is the desired result) that the function be called again with the particular parameter entered as a constant (with the tolerance set to 0) at the value of the hard-clip limit. Additional coordinate transformation functions can solve for the 3 Euler angles which will yield a rotation matrix that is the transpose of the rotation matrix of the 3 input angles (TransposeAngles), which is useful for alternative forms of the conformal transformation. Another function finds an alternate set of parameters (conformalAltSol) consisting of Euler angles, translation terms, and scale $o$, $\phi$, $\kappa$, $T_x$, $T_y$, $T_z$, and $s$ that applies when the inverse form of the 3D conformal transformation utilizes the transpose of the rotation matrix and differencing of the translation terms before, instead of after, matrix multiplication.

Several functions are included in the PT for computing the $3 \times 3$ rotation matrix that is necessary for 3D coordinate transformations and most photogrammetry computations. The rotation matrix can be computed using the Euler convention of omega-phi-kappa (rotationMatrix), azimuth-elevation-roll (rotationMatrixAzElevRoll), and azimuth-tilt-swing (rotationMatrixAzTiltSwing). The functions Australis2PM and PM2Australis compute either omega-phi-kappa or azimuth-elevation-roll angle sets (as used by the program Australis developed at the University of Melbourne) given either set of angles as input. The function rotationMatrixDuality determines an alternate set (duality) of Euler angles ($o$, $\phi$, $\kappa$) that produces the exact same rotation matrix (to within computer round-off error) as the input angles. This function is useful in reducing confusion when comparing resection or calibration results which might yield either set of equivalent angles. It is important to note that the alternate set of angles is not due to the cyclic nature of the angles (which repeat every $2\pi$) since additions of $\pm 2\pi$ actually produce the same angular camera location at each rotation about the axes. Rather the 3 alternate angles rotate the camera to different angles about $X$, $Y$, and $Z$ while establishing the same final orientation of a camera as the input angles. The output alternate angles from the function are restricted to $\pm \pi$ to reduce confusion due to the cyclic nature of the angles. The output angles are either degrees or radians, depending on the specified units of the input angles. Further discussion of this duality property of the rotation matrix can be found in a PE&RS paper entitled "On the Duality of Relative Orientation" by Tian-Yuan Shih, vol. 56 No.9, Sept. 1990, pp. 1281-1283.

A graphic user interface (GUI) function entitled imageObject makes use of the Gaussian object-image relationship between focal length, object distance, and image distance to allow any one of the 3 variables to
be calculated, given values for the other 2. Note that ideally the camera constant will be equal to the image distance if the lens is focused at the value of the object distance. The GUI has edit boxes for each of the 3 variables for entry or for the display of its value after calculation. The desired single variable of interest is determined by selecting its corresponding solve for variable button. The units of the individual variables can be mixed between mm or inch by selection of their corresponding units radio button. Thus the focal length can be in mm while the object distance is in inches before calculating the image distance in either mm or inch depending on which units radio button is selected for image distance. Another pushbutton optionally produces a plot of image distance versus object distance. Mixed units are also allowed for the plot, with the units indicated in the plot axes labels. A matching non-GUI function is also included (imageObject2). This function uses structures for input and output with fields corresponding to focal length, object distance, and image distance. The variable to be solved for is entered in the proper field as [ ], while setting the other two fields to their input value. The returned output structure contains the variable solved for in addition to the two input known values. Note that unlike the GUI, the units for the matching non-GUI function must be consistent and not mixed.

The function MatchIDs matches to within a user-set tolerance correct centroids from one array with correct target IDs of another array (with only approximate centroids). The function is useful for applying the correct target IDs to automatically generated centroid data, given the correct IDs at approximately the same image locations (possibly found manually). This is necessary since the automatically generated data may not have the correct target labels (IDs) needed for further automated image analyses. The two input argument arrays do not need to be the same size and are not limited to 3 column arrays, but the first 3 columns should be correctly ordered (pntID, xpix, ypix). Any target IDs found in one file, but not in the other do not appear in the output matched file. Note that if the absolute difference between centroid doublets is less than the match tolerance then a match is not made. If that occurs for all rows of the input array for a particular target ID, then that target ID does not appear in the output array. It is useful to compare the size (number of rows) of input and output arrays to determine if any target IDs are missing from the output array (for instance, with size(array,1)). The output array contains all the columns of the input centroid array, but with possibly corrected target IDs in column 1. Thus any additional data from the file with the correct centroid locations (such as bounding box data) is echoed through to the output file.

The function centroidMerge provides for the merging of 2 centroid files with the same number of columns. Multiple centroid files can be merged by invoking the merge function with one of the input files being the output of a previous run of the function. The merge function is useful for cases in which the contrast varies significantly across the image so that it may be necessary to determine centroids in segments of the image. Thus one may have several sets of centroid files with possibly overlapping targets with a mixture of target IDs. The function echoes all data from the first input centroid array. Only those targets of the second centroid array that do not overlap those in the first (within the tolerance of the merge function) are passed to the output array. The final output file will then have unique target IDs, but the IDs associated with targets may be as desired.

The function resectionLocalMin determines 3 alternate sets of exterior orientation (which are possible local minima instead of the desired global minimum) for resection on nearly planar objects. For this function, the calibration plate primary lateral dimensions are assumed to be in the X-Y plane with Z \approx constant (representing uniform depth). One of the concerns of nonlinear least squares solutions such as used in space resection is that a local rather than a global minimum may be found (see Appendix). Whether or not a local minimum rather than the global minimum is found is heavily dependent on the initial estimates of the camera coefficients. For cases with very good initial estimates of the exterior orientation of a camera, the global minimum is readily found. However, for cases where it may be necessary to set all the initial estimates to zero (except possibly Zc) it is then found that sometimes a local minimum is found for which the residuals may be comparable or quite a bit larger than the global minimum. For these local minima the exterior
orientation of the camera is incorrect. This effect is especially relevant to wind tunnel and solar sail applications since quite often targets on the object of interest are found to lie almost in a plane. With the alternate sets of exterior orientation found with this function, the local minimum can be transformed to the global minimum, or vice versa (which is useful for testing). Note that the approximations for the locations of the local minima become worse as the optical axis of the camera moves away from being normal to the calibration plate.

List of Functions by Category

CALIBRATION
  camcal_fun
  camcal_fun_1
  cameraConstant
dlt
dlt0
distortSolve
lleast
lleast3
RadiomCali_cheby_fun
RadiomCali_poly_fun
residual_exterior
residual_interior1
residual_interior2

CENTROID PROCESSING
centroidMerge
EpipolarLine_x
EpipolarLine_y
EpipolarRelation_x
EpipolarRelation_y
matchIDs
mm2pixel
pixel2mm

2D COORDINATE TRANSFORMATION
  affine2D
  affine2DLLS
  conformal2D
  conformal2Dinv
  conformal2DLLS
  conformal2DNLLS

3D COORDINATE TRANSFORMATION
  conformal3D
  conformal3Dinv
  conformalAltSol
  conformal3DNLLS

IMAGE PROCESSING
centroid
centroid_cal_fun
clicking_target_fun
displayGrayScale
findBackground
grayScaleDisplay
imagePrelim
location_target1_fun
overlayCentroidsBox
pixelXYselect
roiPolySelect
roiSelect

IMAGING
collinearity
distortApply
distortCorrect
imageObject
imageObject2
XYZ2xy
xyplot

PHOTOGRAMMETRY
intersection
resection
resec
resec3
resecA
resec_ZW
resectionLocalMin
singleView
xy2XYZ
xy2XZ

ROTATION MATRIX
Australis2PM
rotationMatrix
rotationMatrixAzElevRoll
rotationMatrixAzTiltSwing
rotationMatrixDuality
TransposeAngles
PM2Australis

UTILITY
helpPT
loadCamStruct
saveCamStruct
Function Document Pages
Purpose
Affine transformation of 2D coordinates

Syntax
\[
x_{\text{trans}} = \text{affine2D}(x_{\text{in}}, \text{Thetaxy}, Txy, Sxy)
\]

Arguments
\[x_{\text{in}}\]
\[N \times 3\] array of the form below:
\[
\begin{array}{ccc}
pt_1 & x_1 & y_1 \\
pt_2 & x_2 & y_2 \\
. & . & . \\
pt_N & x_N & y_N
\end{array}
\]

\[\text{Thetaxy}\]
2-element row or column vector of rotation angles of the \(x\)- and \(y\)-axis in degrees, + for clockwise rotations

\[Txy\]
translation terms, a row or column vector in \(T_x, T_y\) order (2 \(\times 1\) or 1 \(\times 2\)) of the form: \(Txy = [T_x; T_y]\) or \(Txy = [T_x \ T_y]\); The individual translation terms \(T_x, T_y\) are inserted into a column vector for the matrix calculation within the function.

\[Sxy\]
2-element row or column vector of the \(x\)- and \(y\)-axis scale factors, \(S_x\) and \(S_y\)

Output
\[x_{\text{trans}}\]
\[N \times 3\] array of the form below:
\[
\begin{array}{ccc}
pt_1 & x_1 & y_1 \\
pt_2 & x_2 & y_2 \\
. & . & . \\
pt_N & x_N & y_N
\end{array}
\]

Remarks
The affine transformation does not preserve the shape of a 2D object after transformation. Different scales for each axis as well as non-perpendicularity of the axes are allowed.

Example script
\[\text{affine2DExample.m}\]

Equations
The function \(\text{affine2D}\) represents the following matrix equation for column vector entry of \(x\), \(y\):
\[
\begin{bmatrix}
x_i \\
y_i
\end{bmatrix} = m_P S \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix}
\]
where the pseudo rotation matrix \(m_P\) is given by
and the zero-padded $2 \times 2$ scale matrix is given by

$$m_p = \begin{bmatrix} \cos \theta_x & \sin \theta_y \\ -\sin \theta_x & \cos \theta_y \end{bmatrix}$$

or carrying out the matrix multiplication of $m_p$ and $S$

$$S = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$$

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} S_x \cos \theta_x & S_y \sin \theta_y \\ -S_x \sin \theta_x & S_y \cos \theta_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix}$$

Note that the non-perpendicularity of the axes $\phi$ is given by

$$\phi = \theta_y - \theta_x$$
affine2DLLS

Purpose
linear least squares to determine affine transformation coefficients and estimates of their standard deviation for 2D coordinates

Syntax
[Thetaxy, Txy, Sxy, So] = affine2DLLS(xin, xtrans)

Arguments
xin
N × 3 array of the form below:

\[
\begin{array}{ccc}
pt_1 & x_1 & y_1 \\
pt_2 & x_2 & y_2 \\
\vdots & \vdots & \vdots \\
pt_N & x_N & y_N \\
\end{array}
\]

xtrans
N × 3 array of the form below:

\[
\begin{array}{ccc}
pt_1 & x_1 & y_1 \\
pt_2 & x_2 & y_2 \\
\vdots & \vdots & \vdots \\
pt_N & x_N & y_N \\
\end{array}
\]

Output
Thetaxy
2 × 2 array in which the 1st column contains the rotation angles of the x- and y-axis in degrees, + for CW and the 2nd column contains the least squares estimate of their standard deviations in x, y order

Txy
2 × 2 array in which the 1st column contains the x, y translations Tx, Ty and the 2nd column contains the least squares estimate of their standard deviations, in x, y order

Sxy
2 × 2 array in which the 1st column contains the x- and y-axis scale factors and the 2nd column contains the least squares estimate of their standard deviations, in x, y order

Remarks
The affine transformation does not preserve the shape of a 2D object after transformation. Different scales for each axis as well as non-perpendicularity of the axes are allowed.

Example script
affine2DLLSExample.m

Equations
The function affine2DLLS represents the following matrix equation for column vector entry of x, y:

\[
\begin{bmatrix}
x_i \\
y_i
\end{bmatrix} = \begin{bmatrix}
x \\
y
\end{bmatrix} + \begin{bmatrix}
T_x \\
T_y
\end{bmatrix}
\]

14
where the pseudo rotation matrix $m_p$ is given by

$$m_p = \begin{bmatrix} \cos \theta_x & \sin \theta_y \\ -\sin \theta_x & \cos \theta_y \end{bmatrix}$$

and the zero-padded $2 \times 2$ scale matrix is given by

$$S = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$$

or carrying out the matrix multiplication of $m_p$ and $S$

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} S_x \cos \theta_x & S_y \sin \theta_y \\ -S_x \sin \theta_x & S_y \cos \theta_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix}$$

Note that the non-perpendicularity of the axes $\phi$ is given by

$$\phi = \theta_y - \theta_x$$

With the following substitution

$$a_1 = S_x \cos \theta_x$$
$$a_2 = S_y \sin \theta_y$$
$$b_1 = -S_x \sin \theta_x$$
$$b_2 = S_y \cos \theta_y$$

the affine transformation can be written as the following linear equation

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} a_1 & a_2 \\ b_1 & b_2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix}$$

With this linear form of equations, linear least squares can be used to determine the $a$, $b$, $T_x$, and $T_y$ coefficients resulting in 6 unknowns and 2 equations for each coordinate pair. $N$-coordinate pairs result in $2N$ equations in 6 unknowns. The scale and angular terms can then be found from the $a$ and $b$ coefficients as

$$S_x = \sqrt{a_1^2 + b_1^2}$$
$$S_y = \sqrt{a_2^2 + b_2^2}$$
$$\theta_x = \tan^{-1}\left(-\frac{b_1}{a_1}\right)$$
$$\theta_y = \tan^{-1}\left(\frac{a_2}{b_2}\right)$$
The least squares estimates of the standard deviation of the $a$ and $b$ coefficients can be converted to the scale and angular terms through error propagation of the above 4 equations to yield the next set of 4 equations (after some algebraic manipulations). Note that the angular terms, which are in radians, are converted within the function for output in degrees. Also note that the standard deviations for the translation terms, $T_x$, $T_y$, are found directly, without conversion, from the least squares reduction.

$$
\sigma_{S_x} = \sqrt{\frac{a^2\sigma_{a}^2 + b^2\sigma_{b}^2}{a^2 + b^2}}
$$

$$
\sigma_{S_y} = \sqrt{\frac{a^2\sigma_{a}^2 + b^2\sigma_{b}^2}{a^2 + b^2}}
$$

$$
\sigma_{\theta_x} = \sqrt{\frac{a^2\sigma_{b}^2 + b^2\sigma_{a}^2}{a^2 + b^2}}
$$

$$
\sigma_{\theta_y} = \sqrt{\frac{a^2\sigma_{b}^2 + b^2\sigma_{a}^2}{a^2 + b^2}}
$$
Australis2PM

Purpose
Convert from Australis camera orientation angles to PhotoModeler camera orientation angles $\omega, \phi, \kappa$

Syntax
$\Omega \Phi \kappa = \text{Australis2PM}(\text{Azimuth}, \text{Elevation}, \text{Roll})$

Arguments
Azimuth
angle about $Z$-axis, taken as $+$ for CW rotation; in degrees

Elevation
angle about new $Y$-axis formed after the azimuth rotation, taken as $+$ for CCW; in degrees

Roll
angle about the new $X$-axis formed after the azimuth and Elevation rotations, taken as $+$ for CCW rotation; in degrees

Output
$\Omega \Phi \kappa$
output is a $1 \times 3$ array of angles in the order $\omega, \phi, \kappa$

Remarks
Order of application of angles on input is Azimuth, Elevation, Roll. On output order is $\omega, \phi, \kappa$.

Example script
Australis2PMExample.m

Equations

\[
\begin{align*}
m_{11} &= \sin \alpha \sin \varepsilon \sin \rho + \cos \alpha \cos \rho \\
m_{12} &= -\cos \alpha \sin \varepsilon \sin \rho + \sin \alpha \cos \rho \\
m_{13} &= \cos \varepsilon \sin \rho \\
m_{21} &= \sin \alpha \sin \varepsilon \cos \rho - \cos \alpha \sin \rho \\
m_{22} &= -\cos \alpha \sin \varepsilon \cos \rho - \sin \alpha \sin \rho \\
m_{23} &= \cos \alpha \cos \rho \\
m_{31} &= \sin \alpha \cos \varepsilon \\
m_{32} &= -\cos \alpha \cos \varepsilon \\
m_{33} &= -\sin \varepsilon
\end{align*}
\]

where $\alpha = \text{azimuth}$, $\varepsilon = \text{elevation}$, and $\rho = \text{roll}$.

\[
\omega = \tan^{-1} \left( \frac{-m_{32}}{m_{33}} \right) \\
\phi = \sin^{-1} (m_{31}) \\
\kappa = \tan^{-1} \left( \frac{-m_{23}}{m_{22}} \right)
\]
where $\omega, \phi, \kappa$ equal the Euler angles omega, phi, kappa. Note that the 4-quadrant inverse tangent function $\text{atan2}(y, x)$ is used instead of the 2-quadrant $\text{atan}(y/x)$ (which would have limited computed angles to $\pm 90^\circ$ instead of $\pm 180^\circ$) for the arctangent computations within the function.
camcal_fun

Purpose
Determination of camera orientation parameters based on the interactive use of least squares estimation for the exterior orientation parameters and optimization search scheme for some major interior parameters

Syntax
[orien]=camcal_fun(camformat,approrien,xyimag,xyzobj,corr_no)

Arguments
- **camformat**: 1-column array containing the following camera format data:
  - Number of horizontal pixels
  - Number of vertical pixels
  - Horizontal pixel spacing (mm/pixel)
  - Vertical pixel spacing (mm/pixel)

- **approrien**: 1-column array of the approximate camera orientation parameters, \( (\alpha, \varphi, \kappa, X_c, Y_c, Z_c) \) and \( (c, x_p, y_p, S_h / S_v, K_f, K_z) \)

- **xyimag**: 2-column array of the image coordinates (x, y) of a set of targets in pixels

- **xyzobj**: 3-column array of the object space coordinates \( (X, Y, Z) \) of a set of targets, and the units are consistent with \( (X_c, Y_c, Z_c) \) in inches

- **corr_no**: The iteration number for lens distortion correction, for example, \( \text{corr\_no} = 1 \) for small lens distortion

Output
- **orien**: 1-column array of the improved camera orientation parameters by the optimization method, \( (\alpha, \varphi, \kappa, X_c, Y_c, Z_c) \) and \( (c, x_p, y_p, S_h / S_v, K_f, K_z) \)

Remarks
This function alternatively uses non-linear least squares estimation for the exterior orientation parameters and the Matlab function ‘fminsearch’ for the major interior orientation parameters \( (c, x_p, y_p, S_h / S_v, K_f, K_z) \). The weaker parameters \( (K_z, P_1, P_2, P_3) \) are set at zero in minimization process since the Matlab function ‘fminsearch’ does not give a converged solution when they are included in global minimization along with other parameters.

Example script
camcalExample.m

Equations
The detailed description of the optimization method for camera calibration/orientation is given in the following reference.

Liu, T., Cattafesta, L., Radezsky, R., and Burner, A. W., “Photogrammetry applied to wind tunnel testing”, AIAA J. Vol. 38, No. 6, 2000, pp. 964-971
Purpose
Determination of camera orientation parameters using multiple-step optimization

Syntax
\[
[\text{orien}] = \text{camcal\_fun\_1}(\text{xyimag}, \text{xyzobj}, \text{camformat}, \text{ex\_orien\_0}, \text{in\_orien1\_0}, \text{in\_orien2\_0})
\]

Arguments
- \text{camformat}: 1-column array containing the following camera format data:
  - Number of horizontal pixels
  - Number of vertical pixels
  - Horizontal pixel spacing (mm/pixel)
  - Vertical pixel spacing (mm/pixel)
- \text{ex\_orien\_0}: 1-column array of the approximate exterior orientation parameters, 
  \((\omega, \varphi, \kappa, X_e, Y_e, Z_e)\)
- \text{in\_orien1\_0}: 1-column array of the first subset of the approximate interior orientation parameters,
  \((c, x_p, y_p, S_h / S_s, K_1)\)
- \text{in\_orien2\_0}: 1-column array of the second subset of the approximate interior orientation parameters,
  \((K_2, P_1, P_2)\)
- \text{xyimag}: 2-column array of the image coordinates \((x, y)\) of a set of targets in pixels
- \text{xyzobj}: 3-column array of the object space coordinates \((X, Y, Z)\) of a set of targets, and the units are consistent with \((X_e, Y_e, Z_e)\) in inches

Output
- \text{orien}: 1-column array of the improved camera orientation parameters by the optimization method
  \((\omega, \varphi, \kappa, X_e, Y_e, Z_e)\) and \((c, x_p, y_p, S_h / S_s, K_1, K_2, P_1, P_2)\)

Remarks
This function uses the multiple-step optimization method that alternatively calls the Matlab function ‘fminsearch.m’ for optimization of the exterior orientation parameters \((\omega, \varphi, \kappa, X_e, Y_e, Z_e)\) and some major interior orientation parameters \((c, x_p, y_p, S_h / S_s)\).

After these parameters are given, the weaker parameters \((K_2, P_1, P_2)\) are determined by calling the Matlab function ‘fminsearch.m’ once for additional optimization. This function does not need non-linear least squares estimation in ‘camcal\_fun.m’ that may fails in certain case. However, its accuracy is not high.

Example script
- \text{camcal\_1Example.m}, \text{camcal\_1 Example OV10}
The detailed description of the optimization method for camera calibration/orientation is given in the following reference.

Liu, T., Cattafesta, L., Radezsky, R., and Burner, A. W., “Photogrammetry applied to wind tunnel testing”, AIAA J. Vol. 38, No. 6, 2000, pp. 964-971
cameraConstant

Purpose
Finds camera constant (photogrammetric principal distance) given image data at 2 or more
known Z-displacements of a calibration plate (which can be planar) given camera parameters,
image data, and X, Y, Z object space data

Syntax
\[ c = \text{cameraConstant}(\text{cam}, \text{XYZ}) \]

Arguments
\text{cam}
structure array corresponding to N views of the (known) displaced calibration plate with at
least the following fields:

\text{cam(N).c}
start value for principal distance \( c \) (or camera constant), usually \( \text{mm} \)

\text{cam(N).xp}
x-value of the photogrammetric principal point, usually \( \text{mm} \), but always same units as \( c \).

\text{cam(N).yp}
y-value of the photogrammetric principal point, usually \( \text{mm} \), but always same units as \( c \).

\text{cam(N).omega}
start angle in degrees about X-axis, \( \omega \) taken as + for CCW rotation when viewing down the
axis toward the origin

\text{cam(N).phi}
start angle in degrees about Y-axis, \( \phi \) taken as + for CCW rotation when viewing down the
axis toward the origin

\text{cam(N).kappa}
start angle in degrees about Z-axis, \( \kappa \) taken as + for CCW rotation when viewing down the
axis toward the origin

\text{cam(N).Xc}
start X-coordinate of camera perspective center, always same units as \( \text{XYZ} \) object coordinates

\text{cam(N).Yc}
start Y-coordinate of camera perspective center, always same units as \( \text{XYZ} \) object coordinates

\text{cam(N).Zc}
start Z-coordinate of camera perspective center, always same units as \( \text{XYZ} \) object coordinates;
must accurately reflect the differential displacement in \( Z \).

\text{cam(N).xmm}
M X 3 numeric array containing [pntNum \ x_{\text{mm}} \ y_{\text{mm}}] for M image coordinates seen by the
camera for each view N of the displaced calibration plate

\text{XYZ}
M \times 4 numeric array of the form below (with units same as perspective center location, \( X_c, Y_c, Z_c \)):
Output

c
structure with fields as follows:

c.c
principal distance $c$ (or camera constant), usually $mm$ as found by the function

c.cstd
standard deviation of $c$ computed from least squares

Reference

An improved and less restrictive version of a technique presented in the following reference:

Remarks

The function `cameraConstant` should be a useful complement to optimization for camera calibration. A calibration plate is oriented approximately with its $Z$-axis pointing toward the camera. The plate (or equivalently the camera) is then translated known distances in $Z$. Resections are made at each known displacement with an assumed value of the camera constant (also called photogrammetric principal point). The correct camera constant is approximated by the product of the assumed camera constant and the slope of $Z_c$ from resection versus the known $Z$-displacements. For more than two $Z$-displacements, least squares can be used to determine $c$ and an estimate of its standard deviation. The $Z$-axis of the calibration plate should be aligned approximately with the translation axis. However, resection from within the function determines and partially accounts for any slight angle changes or displacement in $X$ and $Y$ while the plate is being translated. The special case of a single image of a 3-step 54-target calibration plate is also allowed. For this special case all 54 targets must be seen. For this special case a step height of 2 inches is hard-coded into the function. The precision for this special case is much worse than for instance, 3 displacements of a cal plate. This special case option is mainly offered for situations in which it is the only data available. Also note there is only one degree of freedom for the 3-step plate single image case so that the estimate of the standard deviation from least squares is not as reliable as for instance, the case with 5 $Z$-displacements.

Example script

cameraConstantExample.m with input files ‘Sample Files\XYZ3.txt’ and ‘Sample Files\XYZ4.txt’

Equations

The camera constant is found from the following expression

$$c = c_0 \text{slope}$$

where $c_0$ is the current value of the camera constant and $\text{slope}$ is determined by least squares from the following linear relationship

$$Z_c = Z_c \text{slope} + b$$

where $Z_c$ are the input values of the $Z$-locations of the camera’s perspective center at each $Z$-displacement of the calibration plate. These values are passed to the function within the input argument `cam(N).Zc`. The actual values of $Z_c$ passed are not critical (other than serving as
start values for resection). However the difference between the values of $Z_c$ is critical as they partially determine the value of $\text{slope}$. The term $Z_r$ represents the computed values of $Z_c$ returned from the resection function that is called internally within the function. The term $b$ is the y-intercept and is ignored within the function. The function iterates to determine the best estimate of $c$ since the results for the resection function, which is called from within the cameraConstant function, are dependent on the value of $c$ that is passed to it as an input argument.

An estimate of the standard deviation of $c$ is found within the least squares reduction as

$$V = \begin{bmatrix} Z_r & \text{slope} + b - Z_c \\ \vdots & \vdots \\ \vdots & \vdots \end{bmatrix}$$

$$S_o = \sqrt{\frac{V^T V}{df}}$$

$$A = \begin{bmatrix} Z_r & I \\ \vdots & I \\ \vdots & I \end{bmatrix}$$

$$\text{cov} = \left[ A^T A \right]^{-1}$$

$$\text{slope}_\sigma = S_o \sqrt{\text{cov}_{\text{diag}}}$$

$$c_\sigma = c_0 \text{slope}_\sigma$$

where $V$ is a column vector of residuals, $df$ is the degrees of freedom, $S_o$ is the standard deviation of unit weight, $\text{cov}$ is the covariance matrix, $\text{cov}_{\text{diag}}$ represents the diagonal elements of the covariance matrix, and $\text{slope}_\sigma$ and $c_\sigma$ are the estimates of the standard deviation of $\text{slope}$ and $c$. 

24
### centroid

**Purpose**
computes gray scale centroid for a region of interest (roi) of a digital image

**Syntax**

\[
xy = \text{centroid}(\text{img}, x, y, \text{delx}, \text{dely})
\]

\[
xy = \text{centroid}(\text{img}, x, y, \text{delx}, \text{dely}, \text{Gback})
\]

In the first syntax above the gray scale centroid is computed for a region of interest (roi) of the digital image \( \text{img} \), which would normally 1st be loaded from a file with \text{imread}, such as 

\[
\text{img} = \text{imread(FileName)}
\]

where \( \text{fileName} \) is a string variable containing the path (if necessary) and file name where the image resides.

The second syntax adds the optional input argument \( \text{Gback} \)

**Arguments**

- **img**
an array containing an image

- **x**
x-value of centered location in pixels to use for computation of centroid of gray scale

- **y**
y-value of centered location in pixels to use for computation of centroid of gray scale

- **delx**
  half-width of area of pixels to be displayed; full-width = 2 \( \times \) \( \text{delx} \); \( \text{delx} = 8 \) yields a full-width of 16

- **dely**
  half-height of area of pixels to be displayed; full-height = 2 \( \times \) \( \text{dely} \); \( \text{dely} = 8 \) yields a full-height of 16

- **Gback**
on optional input argument to be subtracted from every pixel in the roi before computing the gray scale centroid; usually found with function \text{findBackground}

**Output**

\( xy \)
1 \( \times \) 2 vector containing x- and y-value of gray scale centroid. example:

\[
xy = \\
131.4752 \quad 310.6409
\]

**Example script**

\text{centroidExample.m} with input files ‘Sample Files\image1.tif’, ‘Sample Files\centroids1.txt’, ‘Sample Files\image2.tif’, and ‘Sample Files\centroids2.txt’.

**Remarks**

Use \( \text{img} = \text{imread(fileName)} \) where \( \text{fileName} \) is a string variable containing the path (if necessary) and file name where the image of interest resides. \text{imshow(img)} can be used to put the image for the file in a figure before calling function \text{pixelXYselect} if it is necessary to interactively select the target locations for use in a loop to compute centroids. Note that \text{centroid} only computes 1 centroid at a time and must be invoked from within a loop for gray scale centroids of multiple locations (see \text{centroidExample.m} for example of this). Note that...
the standard designation of horizontal pixel location as x and vertical pixel location as y in the usual (x, y) order can lead to confusion when dealing with matrices which are in (row, column) order since the x-value of the pixel location actually corresponds to columns of the matrix representing the digital image, whereas the y-value corresponds to rows. Thus the matrix in terms of x, y has the order (y, x). To reduce the confusion associated with this ordering, for the functions where it is natural to input arguments in x, y order, the code is written to convert internally to rows and columns for working with the matrices before converting back to (x, y) order for output if necessary.

Equations

\[
\bar{x} = \frac{\sum_i \sum_j iG_{ij}}{\sum_i \sum_j G_{ij}}
\]

\[
\bar{y} = \frac{\sum_i \sum_j jG_{ij}}{\sum_i \sum_j G_{ij}}
\]

where \(\bar{x}\) and \(\bar{y}\) are the location of the centroid in pixels, \(G_{ij}\) is the grey scale at each \((i, j)\) pixel location, \(i\) and \(j\) are the locations in pixels in the x and y directions respectively over some region of interest that is typically very much smaller than the image format. The denominator is simply the sum of the grey scale in the region of interest.
### Purpose
Centroid calculation of a selected image area

### Syntax
```plaintext
[xc, yc] = centroid_cal_fun(A)
```

### Arguments
- **A**: local image area selected

### Output
- **xc, yc**: two-column array (xc, yc) of target centroids in pixels

### Remarks
It is assumed in this function that targets in image have higher intensity than background. For dark targets on lighter background, image should be inverted before the use of this function.

### Called by
- `locating_target1_fun.m`

### Equations
The target centroid \((x_c, y_c)\) is defined as

\[
\begin{align*}
x_c &= \frac{\sum \sum x_i I(x_i, y_i)}{\sum \sum I(x_i, y_i)} \\
y_c &= \frac{\sum \sum y_i I(x_i, y_i)}{\sum \sum I(x_i, y_i)}
\end{align*}
\]

where \(I(x_i, y_i)\) is the gray level on an image. When a target contains only a few pixels and the target contrast is not high, the centroid calculation using the above definition may not be accurate.
**Purpose**
merges 2 centroid arrays with the same number of columns into a single centroid array

**Syntax**
`centMerge = centroidMerge(centA, centB, tol)`

**Arguments**
- **centA**
  at least an $N \times 3$ array ($[\text{pnt xpix ypix} \ldots]$ per row). May have been manually created via mouse or with GUI `imagePrelim`. All rows of `centA` are echoed in output array `centMerge`. (`centA` and `centB` must have same number of columns)
  
  $p_{11}$ $x_1$ $y_2$ ...
  $p_{12}$ $x_2$ $y_2$ ...
  .
  .
  .
  $p_{1N}$ $x_N$ $y_N$ ...

- **centB**
  at least $N \times 3$ array ($[\text{pnt xpix ypix} \ldots]$ per row). Target centroid data from `centB` within location tolerance `tol` are not appended to the output array `centMerge`. Target centroid data from `centB` that is outside tolerance `tol` are appended to `centA`, but with new target IDs that start from the maximum target ID of `centA` + 1. (`centA` and `centB` must have same number of columns)

- **tol**
  tolerance in pixels used for match criteria between centroid doublets in arrays `centA` and `centB`.

**Output**
`centMerge`
$N \times 3$ array ($[\text{pnt xpix ypix} \ldots]$ per row) with all targets (rows) of `centA` and targets (rows) of `centB` that are not approximately located at the same locations as `centA`.

**Remarks**
The function `centroidMerge` is useful for cases in which the contrast varies significantly across the image so that it may be necessary to determine centroids in segments of the image. Thus one may have several sets of centroid files with possibly overlapping targets. The function `centroidMerge` echos all data from the 1st input centroid array `centA`. Only those targets of the 2nd centroid array `centB` that do not overlap those in `centA` (within the tolerance `tol`) are passed to the output array. For multiple centroid arrays one can invoke the function again using the output of a previous run of `centroidMerge` (with partially merged array output).

**Example script**
`centroidMergeExample.m` with input files `Sample Files\centa.txt`, `Sample Files\centb.txt`, `Sample Files\centc.txt`, and `Sample Files\cal1.bmp`
clicking_target_fun

Purpose
Determination of target centroids by clicking high-contrast targets

Syntax
\([xc, yc] = \text{clicking\_target\_fun}(\text{imag, No\_targets, bk\_size\_0})\]

Arguments
\text{imag}
Image name after loading an image file (gray or rgb image)

\text{No\_targets}
total number of targets to be selected

\text{bk\_size\_0}
block size for initial searching a target (such as 10 pixels)

Output
\(xc, yc\)
two-column array \((xc, yc)\) of target centroids in pixels

Remarks
It is assumed in this function that targets in image have higher intensity than background. For dark targets on lighter background, image should be inverted before the use of this function.

Example script
\text{clicking\_targetExample.m}

Equations
The target centroid \((x_c, y_c)\) is defined as

\[
x_c = \frac{\sum \sum x_i I(x_i, y_i)}{\sum \sum I(x_i, y_i)}
\]
\[
y_c = \frac{\sum \sum y_i I(x_i, y_i)}{\sum \sum I(x_i, y_i)}
\]

where \(I(x_i, y_i)\) is the gray level on an image. When a target contains only a few pixels and the target contrast is not high, the centroid calculation using the above definition may not be accurate.
collinearity

Purpose
Creates image coordinates given camera parameters and object coordinates

Syntax
xymm = collinearity(cam, XYZ)

Arguments
cam
structure with fields as follows:
cam.c
principal distance \( c \) (or camera constant), usually \( mm \)
cam.xp
\( x \)-value of the photogrammetric principal point, usually \( mm \), but always same units as \( c \).
cam.yp
\( y \)-value of the photogrammetric principal point, usually \( mm \), but always same units as \( c \).
cam.m
3 × 3 rotation matrix, usually from function rotationMatrix
cam.Xc
\( X \)-coordinate of camera perspective center, always same units as \( XYZ \) object coordinates
cam.Yc
\( Y \)-coordinate of camera perspective center, always same units as \( XYZ \) object coordinates
cam.Zc
\( Z \)-coordinate of camera perspective center, always same units as \( XYZ \) object coordinates

XYZ
filename string for a file (like 'fileName') containing \( N \times 4 \) array or the \( N \times 4 \) array itself. The \( XYZ \) array (or text in file) is of the form below (with units same as perspective center location, \( X_c, Y_c, Z_c \)):

\[
\begin{align*}
pt_1 & : X_1, Y_1, Z_1 \\
pt_2 & : X_2, Y_2, Z_2 \\
\vdots & \\
pt_N & : X_N, Y_N, Z_N
\end{align*}
\]

Output
xymm
output is an \( N \times 3 \) array with point numbers taken from \( XYZ \) array. The output array \( xymm \) is of the form:

\[
\begin{align*}
pt_1 & : x_1, y_1 \\
pt_2 & : x_2, y_2 \\
\vdots & \\
pt_N & : x_N, y_N
\end{align*}
\]
Remarks

The collinearity equations are the most fundamental and important equations in photogrammetry. The collinearity function is very useful for modeling and to create image coordinates for test cases. Note that it is sometimes common to use different units for the photogrammetric principal distance \( c \) and point \((x_p, y_p)\) such as \( \text{mm} \), than are used for the location of the camera perspective point \((X_c, Y_c, Z_c)\) and object coordinates \((X, Y, Z)\), which may be in units of \textit{inches} for example. The units of the image coordinates are always in the same units as \( c \) and \( x_p, y_p \) and are independent of the units used for the location of the perspective center and object coordinates. This mixing of disparate units is permissible due to the ratio of the numerator and denominator of the collinearity equations (see Equations below) since the units of the perspective center location and object coordinates appear in both and cancel each other out. The units of the output image coordinates are then determined entirely from \( c \) (along with \( x_p, y_p \)), which multiplies the ratio of the numerator and denominator.

Example script

collinearityExample.m with input files 'Sample Files\XYZ1.txt' and 'Sample Files\cam1.txt'

Equations

\[
\begin{align*}
x &= x_p - c \left[ \frac{m_{11}(X - X_c) + m_{12}(Y - Y_c) + m_{13}(Z - Z_c)}{m_{11}(X - X_c) + m_{12}(Y - Y_c) + m_{13}(Z - Z_c)} \right] \\
y &= y_p - c \left[ \frac{m_{21}(X - X_c) + m_{22}(Y - Y_c) + m_{23}(Z - Z_c)}{m_{21}(X - X_c) + m_{22}(Y - Y_c) + m_{23}(Z - Z_c)} \right]
\end{align*}
\]
Purpose
Conformal transformation of 2D coordinates

Syntax
xtrans = conformal2D(xin, theta, Txy, s)

Arguments
xin
N × 3 array of the form below:

\[ pt_1 \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \]
\[ pt_2 \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} \]
\[ \vdots \]
\[ pt_N \begin{bmatrix} x_N \\ y_N \end{bmatrix} \]

theta
rotation angle in degrees, positive if clockwise

Txy
translation terms, a row or column vector in Tx, Ty order (2 × 1 or 1 × 2) of the form: Txy = [Tx; Ty] or Txy = [Tx Ty]; The individual translation terms Tx and Ty are inserted into a column vector for the matrix calculation within the function.

s
scalar scale

Output
xtrans
N × 3 array of the form below:

\[ pt_1 \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \]
\[ pt_2 \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} \]
\[ \vdots \]
\[ pt_N \begin{bmatrix} x_N \\ y_N \end{bmatrix} \]

Remarks
The conformal transformation preserves the shape of a 2D object after transformation. This form of the transformation represents the first matrix form in Equations below. By passing the negative of the angle theta (θ) to the function an alternate form of the transform can be invoked (see 2nd form of m below).

Example script
conformal2DExample.m

Equations
The function conformal2D represents the following matrix equation for column vector entry of \( x, y \):

\[
\begin{bmatrix}
  x_i \\
  y_i
\end{bmatrix} = s \begin{bmatrix}
  m & x \\
  m & y
\end{bmatrix} + \begin{bmatrix}
  T_x \\
  T_y
\end{bmatrix}
\]

where
passing negative theta, \( \theta \), to the function is equivalent to applying the transpose of \( m \) in the transformation (equal to the inverse since \( m \) is orthogonal), in which case an alternative form of the conformal transformation is then invoked with rotation matrix \( m \) as follows:

\[
m = \begin{bmatrix}
    \cos \theta & \sin \theta \\
    -\sin \theta & \cos \theta
\end{bmatrix}
\]
conformal2Dinv

**Purpose**
Conformal transformation of 2D coordinates

**Syntax**
\[\text{xtrans} = \text{conformal2Dinv}(\text{xin}, \theta, \text{Txy}, s)\]

**Arguments**
- **xin**
  N × 3 array of the form below:
  \[
  \begin{array}{ccc}
  pt_1 & x_1 & y_1 \\
  pt_2 & x_2 & y_2 \\
  \vdots & \vdots & \vdots \\
  pt_N & x_N & y_N \\
  \end{array}
  \]
- **theta**
  rotation angle in degrees, positive if clockwise
- **Txy**
  translation terms, a row or column vector in Tx, Ty order (2 × 1 or 1 × 2) of the form: \[\text{Txy} = [\text{Tx}; \text{Ty}]\] or \[\text{Txy} = [\text{Tx} \ \text{Ty}]\]; The individual translation terms Tx and Ty are inserted into a column vector for the matrix calculation within the function.
- **s**
  scalar scale

**Output**
- **xtrans**
  N × 3 array of the form below:
  \[
  \begin{array}{ccc}
  pt_1 & x_1 & y_1 \\
  pt_2 & x_2 & y_2 \\
  \vdots & \vdots & \vdots \\
  pt_N & x_N & y_N \\
  \end{array}
  \]

**Remarks**
The conformal transformation preserves the shape of a 2D object after transformation. This form of the transformation represents the first matrix form in **Equations** below. By passing the negative of the angle \(\theta\) to the function an alternate form of the transform can be invoked (see 2nd form of \(m\) below).

**Example script**
conformal2DinvExample.m

**Equations**
The function conformal2Dinv represents the following matrix equation for column vector entry of \(x, y\):
\[
\begin{bmatrix}
  x \\
  y
\end{bmatrix} = s^{-1} m^\top
\begin{bmatrix}
  x - Tx \\
  y - Ty
\end{bmatrix}
\]
where
passing negative theta, \( \theta \), to the function is equivalent to applying the transpose of \( m \) in the transformation above (note that the transpose of \( m \) is equal to the inverse since \( m \) is orthogonal), in which case an alternative form of the conformal transformation is then invoked with rotation matrix \( m \) as follows:

\[
m = \begin{bmatrix}
\cos \theta & \sin \theta \\
-\sin \theta & \cos \theta
\end{bmatrix}
\]
The function \texttt{conformal2DLLS} represents the following matrix equation for column vector entry of \(x, y\):

\[
\begin{bmatrix}
x \\
y
\end{bmatrix} = s \cdot \begin{bmatrix} x \end{bmatrix} + T \begin{bmatrix} T_x \\ T_y \end{bmatrix}
\]

where the rotation matrix \(m\) is given by
entering the terms of the rotation matrix \( m \), the equations become

\[
\begin{pmatrix}
x_i \\
y_i
\end{pmatrix} = \begin{pmatrix}
s \cos \theta & s \sin \theta \\
-s \sin \theta & s \cos \theta
\end{pmatrix} \begin{pmatrix}
x \\
y
\end{pmatrix} + \begin{pmatrix}
T_x \\
T_y
\end{pmatrix}
\]

With the following substitution

\[
a = s \cos \theta \\
b = s \sin \theta
\]

The conformal transformation can be written as the following linear equation

\[
\begin{pmatrix}
x_i \\
y_i
\end{pmatrix} = \begin{pmatrix}
a & b \\
-b & a
\end{pmatrix} \begin{pmatrix}
x \\
y
\end{pmatrix} + \begin{pmatrix}
T_x \\
T_y
\end{pmatrix}
\]

With this linear form of equations, linear least squares can be used to determine the \( a \), \( b \), \( T_x \), and \( T_y \) coefficients resulting in 4 unknowns and 2 equations for each coordinate pair. \( N \)-coordinate pairs results in \( 2N \) equations in 4 unknowns. The scale and angular term can then be found from the \( a \) and \( b \) coefficients as

\[
s = \sqrt{a^2 + b^2}
\]

\[
\theta = \tan^{-1} \left( \frac{b}{a} \right)
\]

The least squares estimates of the standard deviation of the \( a \) and \( b \) coefficients can be converted to the scale and angular terms through error propagation of the above 2 equations to yield (after some algebraic manipulations) the next set of 2 equations. Note that the angular term, which is in radians, is converted within the function for output in degrees. Also note that the standard deviations for the translation terms, \( T_x \), \( T_y \), are found directly, without conversion, from the least squares reduction.

\[
\sigma_s = \sqrt{\frac{a^2 \sigma_a^2 + b^2 \sigma_b^2}{a^2 + b^2}}
\]

\[
\sigma_\theta = \sqrt{\frac{a^2 \sigma_a^2 + b^2 \sigma_b^2}{a^2 + b^2}}
\]
Purpose

non-linear least squares (NLLS) to determine conformal transformation coefficients and estimates of their standard deviation for 2D coordinates

Syntax

Parameter = conformal2DNLLS(xin, xtrans, Start)

Arguments

xin

N × 3 array of the form below:

\[
\begin{align*}
pt_1 & \quad x_1 \quad y_1 \\
pt_2 & \quad x_2 \quad y_2 \\
\vdots & \quad \vdots \\
pt_N & \quad x_N \quad y_N
\end{align*}
\]

xtrans

N × 3 array of the form below:

\[
\begin{align*}
pt_1 & \quad x_1 \quad y_1 \\
pt_2 & \quad x_2 \quad y_2 \\
\vdots & \quad \vdots \\
pt_N & \quad x_N \quad y_N
\end{align*}
\]

Start

input start-value structure with the following fields

Start.theta - start value for theta (degrees)

Start.thetaTol - tolerance for theta within NLLS; for all tolerances [] indicates no tolerance or free to vary, 0 indicates treat the parameter as a constant (do not solve for parameter), and a finite value sets a hard clip range of parameter ± tolerance within the non-linear least squares function

Start.Tx; Start.TxTol translation in x-direction; tolerance

Start.Ty; Start.TyTol translation in y-direction; tolerance

Start.s; Start.sTol scale; tolerance

Output

Parameter

structure with the following fields:

Parameter.theta - rotation angle in degrees, + for CW

Parameter.Tx - x-translation of transformation \(Tx\)

Parameter.Ty - y-translation of transformation \(Ty\)

Parameter.s - scale \(s\)

Parameter.thetastd - estimated standard deviation from NLLS

Parameter.Txstd - estimated standard deviation from NLLS

Parameter.Tystd - estimated standard deviation from NLLS

Parameter.sstd - estimated standard deviation from NLLS

So

scalar which contains the least squares standard deviation of unit weight
Remarks

The conformal transformation preserves the shape of a 2D object after transformation. The non-linear version requires start values for the iterations necessary for the solution. Note that unlike the linear least squares reduction form of the conformal transformation equations found in `conformal2DLLS`, the estimated standard deviations from `conformal2DNLLS` do not require error propagation from the linear $a, b$ coefficients. Also note that this function can selectively solve for any or all of the parameters, $\theta$, $Tx$, $Ty$, $s$, or can use tolerances to limit the variation of those parameters within the non-linear least squares reduction. Note that the hard-clip nature of the tolerances must be used with care since the outputted standard deviations can be misleading. If the outputted parameter is driven to either hard-clip edge, to find out the actual statistics at that value of the parameter the function should be invoked again with the clipped value of the parameter passed as a constant ($\text{Start.parameterTol} = 0$).

Example script

`conformal2DNLLSExample.m`

Equations

The function `conformal2DNLLS` represents the following matrix equation for column vector entry of $x, y$:

$$
\begin{bmatrix}
x_i \\
y_i
\end{bmatrix} = s \cdot m \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix}
$$

where the rotation matrix $m$ is given by

$$
m = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix}
$$

entering the terms of the rotation matrix $m$, the equations become

$$
\begin{bmatrix}
x_i \\
y_i
\end{bmatrix} = \begin{bmatrix} scos \theta & ssin \theta \\ -ssin \theta & scos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix}
$$
Conformal3D

Purpose
Conformal 3D transformation of coordinates

Syntax
\[ X2 = \text{conformal3D}(X1, m, Txyz, s) \]

Arguments
- \( X1 \): \( N \times 4 \) array of the form below:
  
  \[
  pt_1 \quad X_1 \quad Y_1 \quad Z_1 \\
  pt_2 \quad X_2 \quad Y_2 \quad Z_2 \\
  \vdots \\
  pt_N \quad X_N \quad Y_N \quad Z_N
  \]
- \( m \): \( 3 \times 3 \) rotation matrix, usually from function \text{rotationMatrix}
- \( Txyz \): translation terms, a row or column vector in \( X, Y, Z \) order (\( 3 \times 1 \) or \( 1 \times 3 \)) of the form:
  
  \[
  Txyz = [T_x; T_y; T_z] \text{ or } Txyz = [T_x \ T_y \ T_z];
  \]
  The individual translation terms \( T_x, T_y, \) and \( T_z \) are inserted into a column vector for the matrix calculation within the function.
- \( s \): scalar scale

Output
\( X2 \)
\( N \times 4 \) array of the form below:

\[
pt_1 \quad X_1 \quad Y_1 \quad Z_1 \\
pt_2 \quad X_2 \quad Y_2 \quad Z_2 \\
\vdots \\
pt_N \quad X_N \quad Y_N \quad Z_N
\]

Remarks
The conformal transformation preserves the shape of a 3D object after transformation. This form of the transformation represents the first matrix form in \textbf{Equations} below. By passing the transpose of \( m \) to the function an alternate form of the transform can be invoked (see 2\textsuperscript{nd} matrix equation below). The functions \text{conformal3D} and \text{conformal3Dinv} make up a transform pair.

Example script
\text{conformal3DExample.m}

Equations
The function \text{conformal3D} represents the following matrix equation for column vector entry of \( X, Y, Z \):

\[
\begin{bmatrix}
X' \\
Y' \\
Z'
\end{bmatrix} = s \begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix} + \begin{bmatrix}
T_x \\
T_y \\
T_z
\end{bmatrix}
\]
passing the transpose of \( m \) (denoted by \( m' \) in MATLAB) to the function is equivalent to:

\[
\begin{bmatrix}
X_t \\
Y_t \\
Z_t
\end{bmatrix} = s m' 
\begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix} +
\begin{bmatrix}
T_3 \\
T_4 \\
T_5
\end{bmatrix}
\]
Purpose
Inverse conformal 3D transformation of coordinates

Syntax
Xout = conformal3Dinv(Xin, m, Txyz, s)

Arguments
XIN
N × 4 array of the form below:

\[ pt_1 \ X_1 \ Y_1 \ Z_1 \]
\[ pt_2 \ X_2 \ Y_2 \ Z_2 \]
\[ \ldots \]
\[ pt_N \ X_N \ Y_N \ Z_N \]

m
3 × 3 rotation matrix, usually from function rotationMatrix

Txyz
translation terms, a row or column vector in X, Y, Z order (3 × 1 or 1 × 3) of the form: Txyz = [Tx; Ty; Tz] or Txyz = [Tx Ty Tz]; The individual translation terms Tx, Ty, and Tz are inserted into a column vector for the matrix calculation within the function.

s
scalar scale

Output
Xout
N × 4 array of the form below:

\[ pt_1 \ X_1 \ Y_1 \ Z_1 \]
\[ pt_2 \ X_2 \ Y_2 \ Z_2 \]
\[ \ldots \]
\[ pt_N \ X_N \ Y_N \ Z_N \]

Remarks
The inverse conformal transformation preserves the shape of a 3D object after transformation. The functions conformal3D and conformal3Dinv make up a transform pair. This form of the transformation represents the first matrix form in Equations below. By passing the transpose of m to the function an alternate form of the inverse transform can be invoked (see 2nd matrix equation below).

Example script
conformal3DinvExample.m

Equations
The function conformal3Dinv represents the following matrix equation for column vector entry of \( X, \ Y, \ Z \):

\[
\begin{bmatrix}
X_i \\
Y_i \\
Z_i
\end{bmatrix} = s^{-1} m^{-T} \begin{bmatrix}
X - T_x \\
Y - T_y \\
Z - T_z
\end{bmatrix}
\]
passing the transpose of \( m \) (denoted by \( m' \) in MATLAB) to the function is equivalent to:

\[
\begin{bmatrix}
X_t \\
Y_t \\
Z_t
\end{bmatrix} = s^{-1} m \begin{bmatrix}
X - T_x \\
Y - T_y \\
Z - T_z
\end{bmatrix}
\]
Purpose
non-linear least squares (NLLS) to determine conformal transformation coefficients and estimates of their standard deviation for 3D coordinates

Syntax
Parameter = conformal3DNLLS(XYZ1, XYZ2, Start)

Arguments
XYZ1
N × 4 array of the form below:

\[ pt_1 X_1 Y_1 Z_1 \]
\[ pt_2 X_2 Y_2 Z_1 \]
\[ . \]
\[ . \]
\[ pt_N X_N Y_N Z_1 \]

XYZ2
N × 4 array of the form below:

\[ pt_1 X_1 Y_1 Z_1 \]
\[ pt_2 X_2 Y_2 Z_1 \]
\[ . \]
\[ . \]
\[ pt_N X_N Y_N Z_1 \]

Start
input start-value structure with the following fields

Start.omega
start angle \( \omega \) about \( X \), + CCW, degrees

Start.omegaTol
tolerance for \( \omega \) within NLLS; for all tolerances [] indicates no tolerance or free to vary, 0 indicates treat the parameter as a constant (do not solve for parameter), and a finite value sets a hard clip range of parameter \( \pm \) tolerance within the non-linear least squares function

Start.phi
start angle \( \phi \) about \( Y \), + CCW, degrees

Start.phiTol
tolerance for \( \phi \) within NLLS; for all tolerances [] indicates no tolerance or free to vary, 0 indicates treat the parameter as a constant (do not solve for parameter), and a finite value sets a hard clip range of parameter \( \pm \) tolerance within the non-linear least squares function

Start.kappa
start angle \( \kappa \) about \( Z \), + CCW, degrees

Start.kappaTol
tolerance for $\kappa$ within NLLS; for all tolerances [] indicates no tolerance or free to vary, 0 indicates treat the parameter as a constant (do not solve for parameter), and a finite value sets a hard clip range of parameter ± tolerance within the non-linear least squares function

**Start.Tx**
start value for translation in $X$-direction, same units as $XYZ1$ and $XYZ2$

**Start.TxTol**
tolerance for $T_x$ within NLLS; for all tolerances [] indicates no tolerance or free to vary, 0 indicates treat the parameter as a constant (do not solve for parameter), and a finite value sets a hard clip range of parameter ± tolerance within the non-linear least squares function

**Start.Ty**
start value for translation in $Y$-direction, same units as $XYZ1$ and $XYZ2$

**Start.TyTol**
tolerance for $T_y$ within NLLS; for all tolerances [] indicates no tolerance or free to vary, 0 indicates treat the parameter as a constant (do not solve for parameter), and a finite value sets a hard clip range of parameter ± tolerance within the non-linear least squares function

**Start.Tz**
start value for translation in $Z$-direction, same units as $XYZ1$ and $XYZ2$

**Start.TzTol**
tolerance for $T_z$ within NLLS; for all tolerances [] indicates no tolerance or free to vary, 0 indicates treat the parameter as a constant (do not solve for parameter), and a finite value sets a hard clip range of parameter ± tolerance within the non-linear least squares function

**Start.s**
start value for scale $s$

**Start.sTol**
tolerance for $s$ within NLLS; for all tolerances [] indicates no tolerance or free to vary, 0 indicates treat the parameter as a constant (do not solve for parameter), and a finite value sets a hard clip range of parameter ± tolerance within the non-linear least squares function

**Output**
structure with the following fields:

**Parameter.omega**
angle $\omega$ about $X$, + CCW, degrees

**Parameter.phi**
angle $\phi$ about $Y$, + CCW, degrees

**Parameter.kappa**
angle $\kappa$ about $Z$, + CCW, degrees

**Parameter.Tx**
value for translation in $X$-direction, same units as $XYZ1$ and $XYZ2$

**Parameter.Ty**
value for translation in $Y$-direction, same units as $XYZ1$ and $XYZ2$

**Parameter.Tz**
value for translation in Z-direction, same units as XYZ1 and XYZ2

Parameter.s
scale s

Parameter.omegastd
estimated standard deviation from NLLS

Parameter.phistd
estimated standard deviation from NLLS

Parameter.kappastd
estimated standard deviation from NLLS

Parameter.Txstd
estimated standard deviation of $T_x$ from NLLS

Parameter.Tystd
estimated standard deviation of $T_y$ from NLLS

Parameter.Tzstd
estimated standard deviation of $T_z$ from NLLS

Parameter.sstd
estimated standard deviation of s from NLLS

Parameter.So
least squares standard deviation of unit weight

Reference
Elements of Photogrammetry, Paul R. Wolf, 2nd edition, McGraw-Hill, p. 593-596, but modified for the non-transpose form of the 3D conformal transformation

Remarks
The conformal transformation preserves the shape of a 3D object after transformation. The function can be used to selectively solve for any or all of the parameters, omega, phi, kappa, Tx, Ty, Tz, or s, or can use tolerances to limit the variation of those parameters within the non-linear least squares reduction. Note that the hard-clip nature of the tolerances must be used with care since the outputted standard deviations can be misleading. If the outputted parameter is driven to either hard-clip edge, to find out the actual statistics at that value of the parameter the function should be invoked again with the clipped value of the parameter passed as a constant (Start.parameterTol = 0).

Example script
conformal3DNLLSExample.m

Equations
The function conformal3DNLLS represents the following matrix equation for column vector entry of $X$, $Y$, $Z$:

$$
\begin{bmatrix}
X_t \\
Y_t \\
Z_t
\end{bmatrix} = s m \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix}
$$

The function TransposeAngles can be used to establish a new set of $\omega_T$, $\phi_T$, $\kappa_T$ if the form of the conformal transformation is desired which utilizes the transpose of the rotation matrix.
The function `conformal3dNLLS` uses the linearization method (sometimes called the Gauss, Gauss-Newton, or Taylor series method) to solve the non-linear least squares problem. For this method, the 3D conformal equations above are linearized using Taylor’s theorem. This linearization yields 3 equations (1 each for \( X \), \( Y \), and \( Z \) in the 2 coordinate systems) for each 3D point containing initial approximations and products of the partial derivatives and the corrections to be solved for by linear least squares and applied iteratively to the initial approximations. Using the notation of Wolf’s 2\textsuperscript{nd} edition of *Elements of Photogrammetry*, but without using the transpose of the rotation matrix \( m \) to define the 3D conformal transformation, the following matrix equation applies for a single point. The final estimates of the parameters are found from the over-determined set of equations representing all the 3D locations with common target point numbers in both \( XYZ \) data sets (3 equations for each 3D location). Note that the correction terms \( ds, d\phi, d\kappa, dT_x, dT_y, dT_z \) are solved for, not the parameters \( s, a, \phi, \kappa, T_x, T_y, T_z \) themselves. During each iteration of the non-linear least squares the correction terms found by linear least squares are added to the initial start values of each parameter. After several iterations the corrections approach zero and the final iterated solutions for the parameters are determined. To avoid the possibility of an endless loop, the function uses a fixed number of 20 iterations for exit from the function instead of testing for corrections that approach negligibly small values.

\[
\begin{bmatrix}
X - s(m_{11} X + m_{12} Y + m_{13} Z) - T_x \\
Y - s(m_{21} X + m_{22} Y + m_{23} Z) - T_y \\
Z - s(m_{31} X + m_{32} Y + m_{33} Z) - T_z
\end{bmatrix}
= 
\begin{bmatrix}
\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} \\
\end{bmatrix} ds \\
\begin{bmatrix} a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} \\
\end{bmatrix} d\phi \\
\begin{bmatrix} a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} \\
\end{bmatrix} dT_x \\
\begin{bmatrix} ds \\
\end{bmatrix}
\end{bmatrix}
\]

where the \( a \)-terms are given by:

\[
\begin{align*}
a_{11} &= m_{11} X + m_{12} Y + m_{13} Z \\
a_{12} &= s(-m_{11} Y + m_{12} Z) \\
a_{13} &= s(-X \sin \phi \cos \kappa + Y \sin \omega \cos \phi \cos \kappa - Z \cos \omega \cos \phi \cos \kappa) \\
a_{14} &= s(m_{12} X + m_{22} Y + m_{23} Z) \\
a_{15} &= a_{26} = a_{37} = I \\
a_{16} &= a_{17} = a_{25} = a_{27} = a_{34} = a_{35} = a_{36} \\
a_{22} &= m_{21} X + m_{22} Y + m_{23} Z \\
a_{23} &= s(-m_{21} Y + m_{22} Z) \\
a_{24} &= s(X \sin \phi \sin \kappa - Y \sin \omega \cos \phi \sin \kappa + Z \cos \omega \cos \phi \sin \kappa) \\
a_{25} &= s(-m_{11} X - m_{12} Y - m_{13} Z) \\
a_{31} &= m_{31} X + m_{32} Y + m_{33} Z \\
a_{32} &= s(-m_{31} Y + m_{32} Z) \\
a_{33} &= s(X \cos \phi + Y \sin \omega \sin \phi - Z \cos \omega \sin \phi)
\end{align*}
\]
the estimated standard deviations of the correction terms (and hence the parameters themselves) are given by

\[
V = \begin{bmatrix}
-X_i + s \left( m_{1i} X + m_{1j} Y + m_{1j} Z \right) + T_s \\
-Y_i + s \left( m_{2i} X + m_{2j} Y + m_{2j} Z \right) + T_y \\
-Z_i + s \left( m_{3i} X + m_{3j} Y + m_{3j} Z \right) + T_z
\end{bmatrix}
\]

\[
S_0 = \sqrt{\frac{V^T V}{df}}
\]

\[
cov = \left( [A^T] [A] \right)^{-1}
\]

\[
\begin{bmatrix}
s \\
\omega \\
\phi \\
\kappa \\
T_s \\
T_y \\
T_z
\end{bmatrix} = S_0 \sqrt{\text{cov}_{\text{diag}}}
\]

where \( V \) is a column vector of residuals, \( S_0 \) is the standard deviation of unit weight, \( df \) is the degrees of freedom, \( \text{cov} \) is the covariance matrix, \( \text{cov}_{\text{diag}} \) represents the diagonal elements of the covariance matrix, \( A \) is the matrix of \( a \) coefficients, and \( s, \omega, \phi, \kappa, T_s, T_y, T_z \) are the estimates of the standard deviation of \( s, \omega, \phi, \kappa, T_s, T_y, T_z \) from least squares.
ConformalAltSol

Purpose
returns parameters \( \omega_A, \phi_A, \kappa_A, T_{x_A}, T_{y_A}, T_{z_A}, s_A \) for use in the alternate form of the 3D conformal transformation

Syntax
Alternate = ConformalAltSol(Parameter)

Arguments
Parameter
structure with at least the following fields:

Parameter.omega
angle in degrees about X-axis, \( \omega \) taken as + for CCW rotation when viewing down the axis toward the origin

Parameter.phi
angle in degrees about Y-axis, \( \phi \) taken as + for CCW rotation when viewing down the axis toward the origin

Parameter.kappa
angle in degrees about Z-axis, \( \kappa \) taken as + for CCW rotation when viewing down the axis toward the origin

Parameter.Tx
X-translation of transformation \( T_x \)

Parameter.Ty
Y-translation of transformation \( T_y \)

Parameter.Tz
Z-translation of transformation \( T_z \)

Parameter.s
scale \( s \)

Output
Alternate
structure with the following fields:

Alternate.omega
angle in degrees about X-axis, \( \omega_A \) taken as + for CCW rotation when viewing down the axis toward the origin

Alternate.phi
angle in degrees about Y-axis, \( \phi_A \) taken as + for CCW rotation when viewing down the axis toward the origin

Alternate.kappa
angle in degrees about Z-axis, \( \kappa_A \) taken as + for CCW rotation when viewing down the axis toward the origin

Alternate.Tx
X-translation of transformation \( T_{x_A} \)
Alternate.Ty  
Y-translation of transformation $T_{yA}$

Alternate.Tz  
Z-translation of transformation $T_{zA}$

Alternate.s  
scale $s_A$

Remarks  
This function can be useful for cases where the solution is desired in terms of the transpose of the rotation matrix (see second matrix equation below), but the solution in hand is in terms of the rotation matrix without transpose as in the first matrix equation below (for example when using the function `conformal3DNLLS`). The function `TransposeAngles` is used by the function to find the angles $\alpha_A, \phi_A, \kappa_A$.

Example script  
`ConformalAltSolExample.m`

Equations  
The input structure `Parameter` contains the parameters $\omega, \phi, \kappa, T_x, T_y, T_z$ and $s$ that are used in the following form of the 3D conformal coordinate transformation, with $m$ being the rotation matrix formed from the angles $\omega, \phi, \kappa$

$$
\begin{bmatrix}
X_t \\
Y_t \\
Z_t
\end{bmatrix} = s \ m \begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix} + 
\begin{bmatrix}
T_x \\
T_y \\
T_z
\end{bmatrix}
$$

the function `conformalAltSol` can be used to find an alternate set of parameters $\alpha_A, \phi_A, \kappa_A, T_{x_A}, T_{y_A}, T_{z_A}, s_A$ that yield the same output coordinate transformation of $X, Y, Z$ to $X_t, Y_t, Z_t$, but with the following inverse form, with $m_A^T$ being the transpose of the rotation matrix formed from the angles $\omega_A, \phi_A, \kappa_A$

$$
\begin{bmatrix}
X_t \\
Y_t \\
Z_t
\end{bmatrix} = s_A^{-1} m_A^T \begin{bmatrix}
X - T_{x_A} \\
Y - T_{y_A} \\
Z - T_{z_A}
\end{bmatrix}
$$

The relationships between the 2 sets of parameters are

$$
s_A = \frac{1}{s}$$

$$
m_A = m^T$$

$$
\begin{bmatrix}
T_{x_A} \\
T_{y_A} \\
T_{z_A}
\end{bmatrix} = -\frac{1}{s} \ m^T \begin{bmatrix}
T_x \\
T_y \\
T_z
\end{bmatrix}
$$
the output angles \( \omega_A, \phi_A, \kappa_A \) are found with the function \( \text{TransposeAngles} \) that uses the following equations, where the \( m \)-terms are from the rotation matrix formed from the input angles \( \omega, \phi, \kappa \):

\[
\begin{align*}
\phi_A &= \sin^{-1} m_{33} \\
\omega_A &= -\tan^{-1} \left( \frac{m_{23}}{m_{33}} \right) \\
\kappa_A &= -\tan^{-1} \left( \frac{m_{12}}{m_{11}} \right)
\end{align*}
\]
Purpose
displays the gray scale of the selected target location of an image on the MATLAB Command Window

Syntax
displayGrayScale(img, x, y)
displayGrayScale(img, x, y, delx, dely)
displayGrayScale(img, x, y, delx, dely, Gback)

In the 1st simplest calling syntax above the gray scale is displayed for target location selections on the image img, which would normally 1st be loaded from a file with imread, such as img = imread(fileName) where fileName is a string variable containing the path (if necessary) and file name where the image of interest resides. For this 1st syntax delx = dely = 8; Gback = 0 by default. This syntax requires 3 input arguments.

The 2nd syntax has the arguments delx and dely as inputs for a total of 5 input arguments.

The 3rd syntax adds the optional input argument Gback

Arguments

img
an array containing an image

x
x-value of centered location in pixels to use for display of gray scale

y
y-value of centered location in pixels to use for display of gray scale

delx
half-width of area of pixels to be displayed; full-width = 2 x delx; delx = 8 yields a full-width of 16

dely
half-height of area of pixels to be displayed; full-height = 2 x dely; dely = 8 yields a full-height of 16

Gback
gray scale to be subtracted from every pixel in the display area before displaying on the screen

Output
display of gray scale to MATLAB Command Window

Example script
displayGrayScaleExample.m with input files ‘Sample Files\image1.tif’ and ‘Sample Files\image2.tif’.

Remarks
Use img = imread(fileName) where fileName is a string variable containing the path (if necessary) and file name where the image of interest resides. imshow(img) can be used to put the image for the file in a figure before calling function pixelXYselect if it is necessary to interactively select the target locations for display before invoking displayGrayScale. Note that displayGrayScale only displays one area at a time and must be invoked from within a loop for gray scale displays of multiple locations (see displayGrayScaleExample.m for
example of this). Note that the standard designation of horizontal pixel location as \( x \) and vertical pixel location as \( y \) in the usual \((x, y)\) order can lead to confusion when dealing with matrices which are in (row, column) order since the \( x \)-value of the pixel location actually corresponds to columns of the matrix representing the digital image, whereas the \( y \)-value corresponds to rows. Thus the matrix in terms of \( x, y \) has the order \((y, x)\). To reduce the confusion associated with this ordering, for the functions where it is natural to input arguments in \( x, y \) order, the code is written to convert internally to rows and columns for working with the matrices before converting back to \((x, y)\) order for output if necessary.
distortApply

**Purpose**
Applies distortion to image coordinates (in mm)

**Syntax**
\[ \text{xymmDist} = \text{distortApply}(\text{xymm, camDistort}) \]

**Arguments**
- \text{xymm}:
an N × 2 (without target numbers) or N × 3 array with target numbers. If N × 2, target numbers are taken as sequential from 1:Nrows. If \text{xymm} is a character variable representing the name and path to a file, than the array \text{xymm} is loaded from that text file assuming a 2 or 3 column array. The N × 3 version of the array \text{xymm} is of the form:

\[
\begin{align*}
pt_1 & \quad x_1 & \quad y_1 \\
pt_2 & \quad x_2 & \quad y_2 \\
\vdots & \quad \vdots & \quad \vdots \\
pt_N & \quad x_N & \quad y_N
\end{align*}
\]

- \text{camDistort}:
structure with fields as follows:

  - \text{camDistort.x0}:
x-value of point of symmetry for distortion (\(x_s\) in Equations section below), usually mm

  - \text{camDistort.y0}:
y-value of point of symmetry for distortion, (\(y_s\) in Equations section below), usually mm

  - \text{camDistort.K1}:
3\textsuperscript{rd} order radial distortion coefficient (mm\(^{-2}\))

  - \text{camDistort.K2}:
5\textsuperscript{th} order radial distortion coefficient (mm\(^{-4}\))

  - \text{camDistort.K3}:
7\textsuperscript{th} order radial distortion coefficient (mm\(^{-6}\))

  - \text{camDistort.P1}:
decentering distortion term, mm\(^{-1}\)

  - \text{camDistort.P2}:
decentering distortion term, mm\(^{-1}\)

**Output**
- \text{xymmDistort}:
output is an N × 3 array with point numbers taken from \text{xymm} array or sequential from 1:Nrows. The output array \text{xymmDistort} is of the form:

\[
\begin{align*}
pt_1 & \quad x_1 & \quad y_1 \\
pt_2 & \quad x_2 & \quad y_2 \\
\vdots & \quad \vdots & \quad \vdots \\
pt_N & \quad x_N & \quad y_N
\end{align*}
\]
Remarks

Distortion coefficients $K_2$, $K_3$, $P_1$, $P_2$ generally have a much smaller effect on the image than
$K_1$ and are often determined with large relative errors. Thus in some cases it may be prudent
to set these coefficients to 0. If the point of symmetry for distortion can not be found
separately from the photogrammetric principal point $x_p, y_p$ than, as a first estimate, it is
recommended that the point of symmetry be set to $x_p, y_p$. The sign convention for the
distortion coefficients is considered to be a standard (although by no means universal) where
a positive $K_1$ indicated pinchusion (+ distortion) and a negative $K_1$ indicates barrel (-
distortion). This function should be useful in modeling or for creating numerical test cases.

Example script

distortApplyExample.m with input file ‘Sample Files\mm2.txt’

Equations

In the equations below, $x_s, y_s$ locates the point of symmetry for distortion (if unknown use the
photogrammetric principal point $x_p, y_p$), $x$ and $x_d$ represent the undistorted and distorted image
coordinates respectively, $r$ is the magnitude of the radius vector from the point of symmetry to
the undistorted image point $(x, y)$, $\delta r$ is the radial distortion error, and $\delta x$ and $\delta y$ are the
orthogonal components of the radial distortion.

\[
\begin{align*}
  r' &= (x - x_s)^2 + (y - y_s)^2 \\
  \delta r &= K_1 r^2 + K_2 r^4 + K_3 r^6 + \ldots \\
  \delta x &= K_1 (x - x_s) r^2 \\
  \delta y &= K_1 (y - y_s) r^2 \\
  x_d &= x + \delta x \quad \Rightarrow \quad x = x_d - \delta x \\
  y_d &= y + \delta y \quad \Rightarrow \quad y = y_d - \delta y \\
  \delta x &= P_1 (r^2 + 2 x^2) + 2 P_2 x y \\
  \delta y &= P_2 (r^2 + 2 y^2) + 2 P_2 x y
\end{align*}
\]
distortCorrect

**Purpose**
Corrects distorted image coordinates (in *mm*)

**Syntax**
\[ \text{xymmCorr} = \text{distortCorrect}(\text{xymmDist}, \text{camDistort}) \]

**Arguments**
\[ \text{xymmDist} \]
an N × 2 (without target numbers) or N × 3 array with target numbers. If N × 2, target numbers are taken as sequential from 1:Nrows. If \text{xymmDist} is a character variable representing the name and path to a file (or the file name and path itself), than the array \text{xymmDist} is loaded from that text file accommodating either a 2 or 3 column array. The N × 3 version of the array \text{xymmDist} is of the form (the N × 2 version drops the 1st column of target numbers):

\[ \begin{align*}
\text{pt}_1 & \quad x_1 & \quad y_2 \\
\text{pt}_2 & \quad x_2 & \quad y_2 \\
\vdots & \quad \ddots & \quad \ddots \\
\text{pt}_N & \quad x_N & \quad y_N
\end{align*} \]

\[ \text{camDistort} \]
structure with fields as follows:

\[ \text{camDistort.x0} \]
x-value of point of symmetry for distortion (\( x_s \) in Equations section below), usually *mm*

\[ \text{camDistort.y0} \]
y-value of point of symmetry for distortion, (\( y_s \) in Equations section below), usually *mm*

\[ \text{camDistort.K1} \]
3\textsuperscript{rd} order radial distortion coefficient (mm\(^{-2}\))

\[ \text{camDistort.K2} \]
5\textsuperscript{th} order radial distortion coefficient (mm\(^{-4}\))

\[ \text{camDistort.K3} \]
7\textsuperscript{th} order radial distortion coefficient (mm\(^{-6}\))

\[ \text{camDistort.P1} \]
decentering distortion term (mm\(^{-1}\))

\[ \text{camDistort.P2} \]
decentering distortion term (mm\(^{-1}\))

**Output**
\[ \text{xymmCorr} \]
output is an N × 3 array with point numbers taken from \text{xymmDist} array or sequential from 1:Nrows of \text{xymmDist}. The output array \text{xymmCorr} is of the form:

\[ \begin{align*}
\text{pt}_1 & \quad x_1 & \quad y_2 \\
\text{pt}_2 & \quad x_2 & \quad y_2
\end{align*} \]
Remarks

Distortion coefficients $K_2, K_3, P_1, P_2$ generally have a much smaller effect on the image than $K_1$ and are sometimes determined with large relative errors. Thus in some cases it may be prudent to set these coefficients to 0. If the point of symmetry for distortion can not be found separately from the photogrammetric principal point $x_p, y_p$, then, as a first estimate, it is recommended that the point of symmetry be set to $x_p, y_p$. The sign convention for the distortion coefficients is considered to be a standard one (although by no means universal) where a positive $K_1$ indicates pinchusion (+ distortion) and a negative $K_1$ indicates barrel (- distortion). Note that the corrected image coordinates are found by subtracting the $x$- and $y$-components of the distortion from the distorted coordinates (which serve as input to the function). However to correctly compute the components of distortion requires that the undistorted image locations be known. Thus it is necessary to iterate, starting with the assumption that the corrected image coordinates are the same as the input distorted coordinates. At each iteration the estimate of the corrected image coordinates is improved. For the usual range of distortion values, several iterations are typically sufficient. For very large values of distortion and large image areas, many iterations may be needed for very accurate results. The function uses 30 iterations (which still executes nearly instantaneously) to mainly help with large distortion, large image area numerical test cases. This function is the primary function to remove distortion from image coordinates when the distortion coefficients are known.

Example script

distortCorrectExample.m with input file ‘Sample Files\mmDist1.txt’

Equations

In the equations below, $x_s, y_s$ locates the point of symmetry for distortion (if unknown use the photogrammetric principal point $x_p, y_p$), $x$ and $x_d$ represent the undistorted and distorted image coordinates respectively, $r$ is the magnitude of the radius vector from the point of symmetry to the undistorted image point $(x, y)$, $\delta r$ is the radial distortion error, and $\delta x$ and $\delta y$ are the orthogonal components of the radial distortion.

\[
\begin{align*}
    r^2 &= (x - x_s)^2 + (y - y_s)^2 \\
    \delta r &= K_1 r^3 + K_2 r^5 + K_3 r^7 + \ldots \\
    \delta x &= K_1 (x - x_s) r^2 \\
    \delta y &= K_1 (y - y_s) r^2 \\
    x_d &= x + \delta x \quad \Rightarrow \quad x = x_d - \delta x \\
    y_d &= y + \delta y \quad \Rightarrow \quad y = y_d - \delta y \\
    \delta x &= P_1(r^2 + 2x^2) + 2P_2 xy \\
    \delta y &= P_1(r^2 + 2y^2) + 2P_2 xy
\end{align*}
\]
distortSolve

**Purpose**
solves for any or all of the distortion coefficients \( K_1, K_2, K_3, P_1, P_2 \)

**Syntax**
camDistort = distortSolve(cam, camDistortStart, XYZ, Niterations, solve4, useLastResults)

**Arguments**
cam
structure with at least the following fields:

- **cam.c**
  principal distance \( c \) (or camera constant), usually \( mm \)

- **cam.xp**
  \( x \)-value of the photogrammetric principal point, \( xp \), usually \( mm \), but always same units as \( c \).

- **cam.yp**
  \( y \)-value of the photogrammetric principal point, \( yp \), usually \( mm \), but always same units as \( c \).

- **cam.omega**
  angle in degrees about \( X \)-axis, \( \omega \) taken as + for CCW rotation when viewing down the axis toward the origin.

- **cam.phi**
  angle in degrees about \( Y \)-axis, \( \phi \) taken as + for CCW rotation when viewing down the axis toward the origin.

- **cam.kappa**
  angle in degrees about \( Z \)-axis, \( \kappa \) taken as + for CCW rotation when viewing down the axis toward the origin.

- **cam.Xc**
  \( X \)-coordinate of camera perspective center, always same units as \( XYZ \) object coordinates.

- **cam.Yc**
  \( Y \)-coordinate of camera perspective center, always same units as \( XYZ \) object coordinates.

- **cam.Zc**
  \( Z \)-coordinate of camera perspective center, always same units as \( XYZ \) object coordinates.

- **cam.xymm**
  distorted image coordinates as an \( N \times 3 \) numeric array containing \([pntNum \ x_{mm} \ y_{mm}]\) for each target point seen by the camera.

- **camDistortStart**
  structure used for start values of the distortion coefficients and final value of the point of symmetry (represented in the equations section with \( x_s, y_s \)) with at least the following fields:
  (all input values of **camDistortStart** except fields \( x_0 \) and \( y_0 \) are ignored if **useLastResults** = 1 and file **temp4distortSolve.mat** exists in current directory):

  - **camDistortStart.x0**
x-value of point of symmetry for distortion \((x,\text{ in Equations section below})\), usually mm, final value that is echoed in output structure camDistort

camDistortStart.y0
\(y\)-value of point of symmetry for distortion, \((y,\text{ in Equations section below})\), usually mm, final value that is echoed in output structure camDistort

camDistortStart.K1
3\text{rd} order radial distortion coefficient (mm\(^{2}\))

camDistortStart.K2
5\text{th} order radial distortion coefficient (mm\(^{4}\))

camDistortStart.K3
7\text{th} order radial distortion coefficient (mm\(^{6}\))

camDistortStart.P1
decentering distortion term, mm\(^{-1}\)

camDistortStart.P2
decentering distortion term, mm\(^{-1}\)

\textbf{XYZ}
\(N \times 4\) numeric array of the form below (with units same as perspective center location, \(X_{c}, Y_{c}, Z_{c}\)):

\[
\begin{align*}
pt_{1} & \quad X_{1} \quad Y_{1} \quad Z_{1} \\
pt_{2} & \quad X_{2} \quad Y_{2} \quad Z_{2} \\
& \quad \ldots \\
pt_{N} & \quad X_{N} \quad Y_{N} \quad Z_{N}
\end{align*}
\]

\textbf{Niterations}
number of iterations for solution

\textbf{solve4}
input structure with at least the following fields, where field = 1 for solve or = 0 for no solve (coefficient fixed to 0)

\textbf{solve4.K1}
\textbf{solve4.K2}
\textbf{solve4.K3}
\textbf{solve4.P1}
\textbf{solve4.P2}

\textbf{useLastResults}
= 1 to use previous output results from file \texttt{temp4distortSolve.mat} in current folder or = 0 to ignore file

\textbf{Output}
camDistort
structure with at least the following fields:
camDistort.x0
x-value of point of symmetry for distortion (x, in Equations section below), usually mm
(echoed from camDistortStart.x0)

camDistort.y0
y-value of point of symmetry for distortion, (y, in Equations section below), usually mm
(echoed from camDistortStart.x0)

camDistort.K1
3rd order radial distortion coefficient (mm^2)

camDistort.K2
5th order radial distortion coefficient (mm^4)

camDistort.K3
7th order radial distortion coefficient (mm^6)

camDistort.P1
decentering distortion term, mm^{-1}

camDistort.P2
decentering distortion term, mm^{-1}

camDistort.K1std
standard deviation of 3rd order radial distortion coefficient (mm^2) from least squares

camDistort.K2std
standard deviation of 5th order radial distortion coefficient (mm^4) from least squares

camDistort.K3std
standard deviation of 7th order radial distortion coefficient (mm^6) from least squares

camDistort.P1std
standard deviation of decentering distortion term, mm^{-1} from least squares

camDistort.P2std
standard deviation of decentering distortion term, mm^{-1} from least squares

camDistort.So
standard deviation of unit weight from least squares

output structure camDistort is written to file temp4distortSolve.mat; use
camDistortSolution = load('temp4distortSolve') to access from MATLAB

Remarks

Distortion coefficients $K_2$, $K_3$, $P_1$, $P_2$ generally have a much smaller effect on the image than $K_1$ and are often determined with large relative errors. Thus in some cases it may be prudent to set these coefficients to 0 and solve only for $K_1$ (all solve4 fields = 0 except for solve4.K1 which should be set to 1). If the point of symmetry for distortion can not be found separately from the photogrammetric principal point $x_p$, $y_p$, then as a first estimate it is recommended that the point of symmetry be set to $x_p$, $y_p$. More reliable solutions are generally obtained with the
camera image plane approximately parallel to the object field, which can be planar. The sign convention for the distortion coefficients is considered to be a standard (although by no means universal) where a positive $K_1$ indicates pinchusion (+ distortion) and a negative $K_1$ indicates barrel (- distortion). The designated distortion coefficients in the structure solve4 are found iteratively by invoking the resection function to determine improved estimates of the exterior orientation parameters $\alpha$, $\phi$, $\kappa$, $X_c$, $Y_c$, and $Z_c$, which are then used in the function collinearity to generate ideal undistorted image coordinates based on the updated exterior orientation and the input object coordinates XYZ. The original inputted distorted image coordinates are then compared to the newly updated estimates of the undistorted image coordinates. The designated distortion coefficients are found by linear least squares. The improved estimates of the distortion coefficients are then used to correct the original distorted image coordinates to create a new set of undistorted image coordinates. The process is repeated for $N$ iterations. It is sometimes necessary to utilize several hundred iterations to converge. Only target point numbers common to both XYZ and cam.xymm are used in the solution.

**Example script**

distortSolveExample.m with input file 'Sample Files\ XYZ3.txt'

**Equations**

In the equations below, $x_s, y_s$ locates the point of symmetry for distortion (if unknown use the photogrammetric principal point $x_p, y_p$), $x$ and $x_d$ represent the undistorted and distorted image coordinates respectively, $r$ is the magnitude of the radius vector from the point of symmetry to the undistorted image point $(x, y)$, $\delta r$ is the radial distortion error, and $\delta x$ and $\delta y$ are the orthogonal components of the radial distortion.

\[
\begin{align*}
    r^2 &= (x-x_s)^2 + (y-y_s)^2 \\
    \delta r &= K_1 r^3 + K_2 r^5 + K_3 r^7 + \ldots \\
    \delta x &= K_1 (x-x_s)r^2 \\
    \delta y &= K_1 (y-y_s)r^2 \\
    x_d &= x + \delta x \quad \Rightarrow \quad x = x_d - \delta x \\
    y_d &= y + \delta y \quad \Rightarrow \quad y = y_d - \delta y \\
    \delta x &= P_1 (r^3 + 2x^2) + 2P_2 xy \\
    \delta y &= P_1 (r^3 + 2y^2) + 2P_2 xy
\end{align*}
\]

The matrices $L$ and $A$ below are built up for each common set of image and object coordinates. The variables $x_d$ and $y_d$ are taken from the input argument field cam.xymm. The variables $x$ and $y$ are the iterated values of the estimates of the undistorted image coordinates. The ellipsis symbols … in the matrices below indicate that the matrices $L$ and $A$ are populated with 2 rows for each target point and may each have many rows. For instance, 54 target points would lead to $L$ and $A$ matrices with 108 rows each. The number of columns of matrix $A$ is dictated by the number of unknown distortion coefficients carried in the solution. The 5 columns of matrix $A$ below represent in order $K_1, K_2, K_3, P_1, P_2$. A column would be missing from matrix $A$ for each coefficient not solved for.
The solution vector Solution is then found by the MATLAB least squares operator ‘\’, where like the matrix A, the number of rows of the solution vector are dictated by the number of unknown distortion coefficients solved for as indicated in the structure solve4.

\[
L = \begin{bmatrix}
    x_d - x \\
    y_d - y \\
    \vdots \\
\end{bmatrix}
\]
\[
A = \begin{bmatrix}
    r_i (x - x_i) & r_i (x - x_i) & r_i (x - x_i) & r_i^2 + 2(x - x_i)^2 & 2(x - x_i)(y - y_i) \\
    r_i (y - y_i) & r_i (y - y_i) & r_i (y - y_i) & 2(x - x_i)(y - y_i) & r_i^2 + 2(y - y_i)^2 \\
    \vdots & \vdots & \vdots & \vdots & \vdots \\
\end{bmatrix}
\]

\[
\text{Solution} = A \backslash L
\]

The estimates of the standard deviation of the coefficients is then found from the following relationships

\[
V = A \text{Solution} - L
\]
\[
S_0 = \sqrt{\frac{V^T V}{df}}
\]
\[
\text{cov} = \begin{bmatrix} A^T A \end{bmatrix}^{-1}
\]
\[
\begin{bmatrix}
    K_{1\sigma} \\
    K_{2\sigma} \\
    K_{3\sigma} \\
    P_{1\sigma} \\
    P_{2\sigma}
\end{bmatrix} = S_0 \sqrt{\text{cov}_{\text{diag}}}
\]

where \( V \) is a column vector of residuals, \( S_0 \) is the standard deviation of unit weight, \( df \) is the degrees of freedom which equals 2 times the number of target points minus the number of coefficients solved for, \( \text{cov} \) is the covariance matrix, \( \text{cov}_{\text{diag}} \) represents the diagonal elements of the covariance matrix, and \( K_{1\sigma}, K_{2\sigma}, K_{3\sigma}, P_{1\sigma}, \) and \( P_{2\sigma} \) are the estimates of the standard deviations of \( K_1, K_2, K_3, P_1, \) and \( P_2 \) from least squares.
Purpose
Approximate estimation of the exterior orientation parameters and principal distance by the raw DLT

Syntax
[L,orien]=dlt0(camformat,xyimag,xyzobj)

Arguments
camformat
1-column array containing the following camera format data:
- Number of horizontal pixels
- Number of vertical pixels
- Horizontal pixel spacing (mm/pixel)
- Vertical pixel spacing (mm/pixel)

xyimag
2-column array of the image coordinates (x, y) of a set of targets in pixels

xyzobj
3-column array of the object space coordinates (X, Y, Z) of a set of targets, and the units are consistent with (X, Y, Z) in inches

Output
L
The DLT parameters

orien
1-column array of the estimated camera orientation parameters \( (\omega, \varphi, \kappa, X_c, Y_c, Z_c) \) and \( (c, 0, 0, S_h / S_v, 0) \)

Remarks
Unlike 'dlt.m', the function ‘dlt0.m’ is the raw DLT where the Euler rotational angles are not converted to the ranges \(-\pi \leq \omega \leq \pi\), \(-\pi / 2 \leq \phi \leq \pi / 2\), and \(-\pi \leq \kappa \leq \pi\). The principal point location \((x_p, y_p)\) and the first radial lens distortion parameter \(K_1\) are set at zero since these parameters given by the DLT are not accurate and very sensitive to lens distortion. The results given by ‘dlt0.m’ are as good as those given by ‘dlt.m’ since the selection of the Euler angles are not refined.

There is a pitfall:
When the image plane is almost parallel to the \((Y, Z)\) plane and the object-space coordinate system and the image coordinate system are transformed through either roughly 90-deg rotation or no rotation, it is found that \(\omega\) is about 90 deg such that \(\tan \omega\) is almost infinite. Therefore, the DLT often has a numerical error in inverting \(\tan \omega\) and cannot automatically provide a correct initial approximation for refinement by the optimization method.

Example script
camcal_funExample.m

Equations
The Direct Linear Transformation (DLT) can be very useful to determine approximate values of the camera parameters. Rearranging the terms in the collinearity equations leads to the DLT equations...
The DLT parameters \( L_1, \ldots, L_{11} \) are related to the camera exterior and interior orientation parameters \((\alpha, \phi, \kappa, X, Y, Z, c)\) and \((c, x_p, y_p)\) (McGlone 1989). Unlike the standard collinearity equations, Eq. (1) is linear for the DLT parameters when the lens distortion terms \(dx\) and \(dy\) are neglected. In fact, the DLT is a linear treatment of what is essentially a non-linear problem at the cost of introducing two additional parameters. The matrix form of the linear DLT equations for \( M \) targets is

\[
\begin{pmatrix}
  x_p \\
  y_p \\
  c \\
  \phi \\
  \omega \\
  \kappa \\
  m_{1j} \\
  L_j \\
  L_0 \\
  L_1 \\
  L_2 \\
  L_3 \\
  L_4 \\
  L_5 \\
  L_6 \\
  L_7 \\
  L_8 \\
  L_9 \\
  L_{10} \\
  L_{11}
\end{pmatrix} = \begin{pmatrix}
  L_1 & L_2 & L_3 & L_4 & L_5 & L_6 & L_7 & L_8 & L_9 & L_{10} & L_{11} & 1
\end{pmatrix}^T
\]

Because of its simplicity, the DLT is widely used in both non-topographic photogrammetry and computer vision. When \(dx\) and \(dy\) cannot be ignored, however, iterative solution methods are still needed and the DLT loses its simplicity. In general, the DLT can be used to obtain fairly good values of the exterior orientation parameter and the principal distance, although it gives a poor estimate for the principal-point location \((x_p, y_p)\). Therefore, the DLT is valuable since it can provide initial approximations for more accurate methods like the optimization method discussed below for comprehensive camera calibration.

Liu, T., Cattafesta, L., Radezsky, R., and Burner, A. W., “Photogrammetry applied to wind tunnel testing”, AIAA J. Vol. 38, No. 6, 2000, pp. 964-971

**Purpose**
Approximate estimation of the exterior orientation parameters and principal distance

**Syntax**

\[
[\text{orien}]=\text{dlt}(\text{camformat}, \text{xyimag}, \text{xyzobj})
\]

**Arguments**

- **camformat**
  1-column array containing the following camera format data:
  - Number of horizontal pixels
  - Number of vertical pixels
  - Horizontal pixel spacing (mm/pixel)
  - Vertical pixel spacing (mm/pixel)

- **xyimag**
  2-column array of the image coordinates (x, y) of a set of targets in pixels

- **xyzobj**
  3-column array of the object space coordinates \((X, Y, Z)\) of a set of targets, and the units are consistent with \((X_c, Y_c, Z_c)\) in inches

**Output**

- **orien**
  1-column array of the estimated camera orientation parameters \((\omega, \varphi, \kappa, X_c, Y_c, Z_c)\) and \((c, 0, 0, S_h / S_v, 0)\)

**Remarks**
In this function, the principal point location \((x_p, y_p)\) and the first radial lens distortion parameter \(K_1\) are set at zero since these parameters given by the DLT are not accurate and very sensitive to lens distortion. The results given by the DLT are good enough as the initial approximation for a more accurate method like the optimization method.

There is a pitfall:
When the image plane is almost parallel to the \((Y, Z)\) plane and the object-space coordinate system and the image coordinate system are transformed through either roughly 90-deg rotation or no rotation, it is found that \(\omega\) is about 90 deg such that \(\tan\omega\) is almost infinite. Therefore, the DLT often has a numerical error in inverting \(\tan\omega\) and cannot automatically provide a correct initial approximation for refinement by the optimization method.

**Example script**
camcalExample.m

**Equations**
The Direct Linear Transformation (DLT) can be very useful to determine approximate values of the camera parameters. Rearranging the terms in the collinearity equations leads to the DLT equations

\[
\begin{align*}
L_1X + L_2Y + L_3Z + L_4-(x+dx)(L_9X + L_{10}Y + L_{11}Z + 1) &= 0 \\
L_2X + L_3Y + L_4Z + L_5-(y+dy)(L_9X + L_{10}Y + L_{11}Z + 1) &= 0
\end{align*}
\]

(1)
The DLT parameters \( L_1, \cdots, L_{11} \) are related to the camera exterior and interior orientation parameters \((x, \phi, \kappa, X, Y, Z, c, x_p, y_p)\) (McGlone 1989). Unlike the standard collinearity equations, Eq. (1) is linear for the DLT parameters when the lens distortion terms \(dx\) and \(dy\) are neglected. In fact, the DLT is a linear treatment of what is essentially a non-linear problem at the cost of introducing two additional parameters. The matrix form of the linear DLT equations for \( M \) targets is \( B L = C \), where \( L = (L_1, \cdots, L_{11})^T \), \( C = (x_i, y_i, \cdots, x_M, y_M)^T \), and \( B \) is the \( 2M \times 11 \) configuration matrix that can be directly obtained from Eq. (1). A least-squares solution for \( L \) is formally given by \( L = (B^T B)^{-1} B^T C \) without using an initial guess. The camera parameters can be extracted from the DLT parameters from the following expressions:

\[
\begin{align*}
L_1^2 + L_2^2 + L_3^2 - x_p^2, \\
\phi = \sin^{-1}(L_2 L_3), \\
\omega = \tan^{-1}(-L_{10}/L_{11}), \\
\kappa = \cos^{-1}(m_{11}/\cos(\phi)), \\
m_{11} = L(x_p L_0 - L_i)c, \\
L = -(L_0^2 + L_{10}^2 + L_{11}^2)^{1/2}, \\
x_p = (L_1 L_0 + L_2 L_{10} + L_3 L_{11})L_2, \\
y_p = (L_1 L_0 + L_2 L_{10} + L_3 L_{11})L_3, \\
c = \sqrt{(L_3^2 + L_4^2 + L_5^2)L_2 - x_p^2}, \\
\theta = \tan^{-1}(L_{10}/L_{11}), \\
\tau = \sin^{-1}(L_2 L_3), \\
\rho = \tan^{-1}(-L_{10}/L_{11}),
\end{align*}
\]

Because of its simplicity, the DLT is widely used in both non-topographic photogrammetry and computer vision. When \(dx\) and \(dy\) cannot be ignored, however, iterative solution methods are still needed and the DLT loses its simplicity. In general, the DLT can be used to obtain fairly good values of the exterior orientation parameter and the principal distance, although it gives a poor estimate for the principal-point location \((x_p, y_p)\). Therefore, the DLT is valuable since it can provide initial approximations for more accurate methods like the optimization method discussed below for comprehensive camera calibration.

Liu, T., Cattafesta, L., Radezsky, R., and Burner, A. W., “Photogrammetry applied to wind tunnel testing”, AIAA J. Vol. 38, No. 6, 2000, pp. 964-971

EpipolarLine_x

Purpose
Determination of the epipolar line in image 1 for a given point in image 2 based on minimization along the x-axis.

Syntax
```
[x_epipo1,y_epipo1,normG_x]=
EpipolarLine_x(ximag2,yimag2,orientation1,orientation2,camformat1,camformat2,x_bound0,x_bound1)
```

Arguments
- **camformat1**: 1-column array containing the camera format data for camera 1:
  - Number of horizontal pixels
  - Number of vertical pixels
  - Horizontal pixel spacing (mm/pixel)
  - Vertical pixel spacing (mm/pixel)

- **camformat2**: 1-column array containing the camera format data for camera 2:
  - Number of horizontal pixels
  - Number of vertical pixels
  - Horizontal pixel spacing (mm/pixel)
  - Vertical pixel spacing (mm/pixel)

- **orientation1**: 1-column array of the camera orientation parameters for camera 1
  - \( \omega, \varphi, \kappa, X_c, Y_c, Z_c \) and \( c, x_p, y_p, S_h / S_v, K_1, K_2, P_1, P_2 \)

- **orientation2**: 1-column array of the camera orientation parameters for camera 2
  - \( \omega, \varphi, \kappa, X_c, Y_c, Z_c \) and \( c, x_p, y_p, S_h / S_v, K_1, K_2, P_1, P_2 \)

- **(ximag2,yimag2)**: image coordinates \((x,y)\) in pixels in image 2

Output
- **(x_epipo1,y_epipo1)**: the coordinates of the epipolar line in image 1
- **normG**: norm of G, where \( \text{norm}(G) = 0 \) on the epipolar line

Remarks
This function uses 'EpipolarRelation_x.m' for minimization along the x-axis in image 1. This function is feasible for an epipolar line that is not vertical in the \((x,y)\) image plane.

Example script
- EpipolarExample.m

Equations
The detailed description of determining an epipolar line is given in the following reference.
**EpipolarLine_y**

**Purpose**
Determination of the epipolar line in image 1 for a given point in image 2 based on minimization along the y-axis.

**Syntax**

```matlab
[x_epipo1,y_epipo1,normG_y] = EpipolarLine_y(ximag2,yimag2,orientation1,orientation2,camformat1,camformat2,y_bound0,y_bound1)
```

**Arguments**

camformat1
1-column array containing the camera format data for camera 1:
- Number of horizontal pixels
- Number of vertical pixels
- Horizontal pixel spacing (mm/pixel)
- Vertical pixel spacing (mm/pixel)

Camformat2
1-column array containing the camera format data for camera 2:
- Number of horizontal pixels
- Number of vertical pixels
- Horizontal pixel spacing (mm/pixel)
- Vertical pixel spacing (mm/pixel)

orientation1
1-column array of the camera orientation parameters for camera 1

- \(\omega, \varphi, \kappa, X_c, Y_c, Z_c\)
- \((c, x_p, y_p, S_h / S_v, K_1, K_2, P_1, P_2)\)

orientation2
1-column array of the camera orientation parameters for camera 2

- \(\omega, \varphi, \kappa, X_c, Y_c, Z_c\)
- \((c, x_p, y_p, S_h / S_v, K_1, K_2, P_1, P_2)\)

(ximag2,yimag2)
image coordinates (x,y) in pixels in image 2

**Output**

- \((x_{epipo1}, y_{epipo1})\)
  the coordinates of the epipolar line in image 1

- \(\text{normG}\)
  norm of G, where \(\text{norm}(G) = 0\) on the epipolar line

**Remarks**
This function uses 'EpipolarRelation_y.m' for minimization along the y-axis in image 1. This function is particularly feasible for an epipolar line that is almost vertical in the (x,y) image plane.

**Example script**
EpipolarExample.m

**Equations**
The detailed description of determining an epipolar line is given in the following reference.
EpipolarRelation_x

Purpose
Calculation of difference norm of the epipolar relation as a function of the x-coordinate in image A for minimization

Syntax
normG = 
EpipolarRelation_x(xPixA, yPixA, xPixB, yPixB, oriA, oriB, camformatA, camformatB)

Arguments

- **camformatA**: 1-column array containing the camera format data for camera A:
  - Number of horizontal pixels
  - Number of vertical pixels
  - Horizontal pixel spacing (mm/pixel)
  - Vertical pixel spacing (mm/pixel)

- **camformatB**: 1-column array containing the camera format data for camera B:
  - Number of horizontal pixels
  - Number of vertical pixels
  - Horizontal pixel spacing (mm/pixel)
  - Vertical pixel spacing (mm/pixel)

- **oriA**: 1-column array of the camera orientation parameters for camera A
  - \( (\phi, \theta, \kappa, X_c, Y_c, Z_c) \) and \( (c, x_p, y_p, S_h / S_v, K_1, K_2, P_1, P_2) \)

- **oriB**: 1-column array of the camera orientation parameters for camera B
  - \( (\phi, \theta, \kappa, X_c, Y_c, Z_c) \) and \( (c, x_p, y_p, S_h / S_v, K_1, K_2, P_1, P_2) \)

- \((xPixA, yPixA)\)
  image coordinates \((x, y)\) in pixels in image A

- \((xPixB, yPixB)\)
  image coordinates \((x, y)\) in pixels in image B

Output
normG
norm of \( G \), where \( \text{norm}(G) = 0 \) on an epipolar line

Remarks
This function is used in 'EpipolarLine_x.m' for minimization along the x-axis in image 1.
This function is feasible for an epipolar line that is not vertical in the \((x, y)\) image plane. In most cases, 'EpipolarLine_x.m' and 'EpipolarLine_y.m' will give the same results.

Example script
EpipolarExample.m

Equations
The detailed description of determining an epipolar line is given in the following reference.
## EpipolarRelation_y

### Purpose
Calculation of difference norm of the epipolar relation as a function of the y-coordinate in image A for minimization

### Syntax

\[
\text{normG} = \text{EpipolarRelation}_y(x\text{PixA},y\text{PixA},x\text{PixB},y\text{PixB},\text{oriA},\text{oriB},\text{camformatA},\text{camformatB})
\]

### Arguments

- **camformatA**: 1-column array containing the camera format data for camera A:
  - Number of horizontal pixels
  - Number of vertical pixels
  - Horizontal pixel spacing (mm/pixel)
  - Vertical pixel spacing (mm/pixel)

- **camformatB**: 1-column array containing the camera format data for camera B:
  - Number of horizontal pixels
  - Number of vertical pixels
  - Horizontal pixel spacing (mm/pixel)
  - Vertical pixel spacing (mm/pixel)

- **oriA**: 1-column array of the camera orientation parameters for camera A
  \((\omega, \varphi, \kappa, X_c, Y_c, Z_c)\) and \((c, x_p, y_p, S_h / S_v, K_1, K_2, P_1, P_2)\)

- **oriB**: 1-column array of the camera orientation parameters for camera B
  \((\omega, \varphi, \kappa, X_c, Y_c, Z_c)\) and \((c, x_p, y_p, S_h / S_v, K_1, K_2, P_1, P_2)\)

- \((x\text{PixA},y\text{PixA})\): image coordinates \((x,y)\) in pixels in image A

- \((x\text{PixB},y\text{PixB})\): image coordinates \((x,y)\) in pixels in image B

### Output

- **normG**: norm of \(G\), where \(\text{norm}(G) = 0\) on an epipolar line

### Remarks
This function is used in 'EpipolarLine_y.m' for minimization along the x-axis in image 1. This function is useful for an epipolar line that is almost vertical in the \((x,y)\) image plane. In that case, 'EpipolarLine_x.m' does not work well. In most cases, 'EpipolarLine_x.m' and 'EpipolarLine_y.m' will give the same results.

### Example script

- EpipolarExample.m

### Equations
The detailed description of determining an epipolar line is given in the following reference.
findBackground

**Purpose**
finds the perimeter max background for a given region of interest (roi) of a digital image

**Syntax**
Gback = findBackground(img, x, y, delx, dely)

The digital image img would normally first be loaded from a file with imread, such as img = imread(fileName) where fileName is a string variable containing the path (if necessary) and file name where the image resides

**Arguments**
- **img**
an array containing an image
- **x**
x-value of centered location in pixels to use for roi
- **y**
y-value of centered location in pixels to use for roi
- **delx**
half-width of area of pixels of roi; full-width = 2 × delx; delx = 8 yields a full-width of 16
- **dely**
half-height of area of pixels of roi; full-height = 2 × dely; dely = 8 yields a full-height of 16

**Output**
Gback
perimeter max background of the region of interest (roi) of a digital image

**Example script**
findBackgroundExample.m with input file ‘Sample Files\image3.tif’.

**Remarks**
Use img = imread(fileName) where fileName is a string variable containing the path (if necessary) and file name where the image of interest resides. imshow(img) can be used to put the image for the file in a figure before then calling function pixelXYselect if it is necessary to interactively select the target locations. Note that findBackground only finds the perimeter background for one roi at a time and must be invoked from within a loop for gray scale displays of multiple locations (see findBackgroundExample.m for example of this). Typically the returned value for Gback is subtracted from a given roi before centroiding to remove the bias error in centroiding that can be caused by background gray scale. It would be prudent to test for the magnitude of Gback to determine if too large a value is being returned for subtraction (possibly indicating that the perimeter of the roi is too close to the target blob). In such a case the roi may need to be enlarged slightly and possibly recentered to improve results. Note that the standard designation of horizontal pixel location as x and vertical pixel location as y in the usual (x, y) order can lead to confusion when dealing with matrices which are in (row, column) order since the x-value of the pixel location actually corresponds to columns of the matrix representing the digital image, whereas the y-value corresponds to rows. Thus the matrix in terms of x, y has the order (y, x). To reduce the confusion associated with this ordering, for the functions where it is natural to input arguments in x, y order, the code is written to convert internally to rows and columns for working with the matrices before converting back to (x, y) order for output if necessary.
# grayScaleDisplay

**Purpose**  
grayscale display (interactive) with image in a figure window

**Syntax**  
grayScaleDisplay(img)

**Arguments**  
img  
image variable in the workspace or a valid image file name (either a character string or character variable); Note that if the function is called without an input argument, an image file dialog box opens from which the user can select the proper image file.

**Output**  
figure window with the image in upper half and the interactive Pixel Region tool in the lower half; X, Y pixel and intensity are shown as the cursor is moved either in the image or in the Pixel Region tool area; slider bars on the Pixel Region tool allow for movement about the image to examine grayscale; a small rectangular box overlay that represents the coverage of the Pixel Region tool can also be moved around the image to change the area of the image that the Pixel Region tool covers.

**Example script**  
grayScaleDisplayExample.m with input files ‘Sample Files\image1.tif’ and ‘Sample Files\image2.tif’

**Remarks**  
every time the function grayScaleDisplay is invoked a new figure window is created. To remove the currently selected figure window, enter ‘close’ at the command line, or ‘close all’ to close all MATLAB figures.
imageObject2

Purpose
Solves for focal length, object distance, or image distance, given any 2 of the 3 parameters (simplified non-GUI version of imageObject)

Syntax
\[
camOut = \text{imageObject2}(camIn)
\]

Arguments
\[
camIn = \text{structure with the following fields:}
\]
\[
camIn.f - \text{focal length } f
\]
\[
camIn.obj - \text{object distance } obj
\]
\[
camIn.img - \text{image distance } img
\]
The variable to be solved for should be set to [ ].

Output
\[
camOut = \text{structure with the following fields, where one of the variables is calculated and 2 of the variables are echoed from the input argument structure } camIn:
\]
\[
camOut.f - \text{focal length } f
\]
\[
camOut.obj - \text{object distance } obj
\]
\[
camOut.img - \text{image distance } img
\]

Remarks
The function imageObject2 uses the Gaussian object-image relationship to determine any 1 of the 3 variables focal length \( f \), object distance \( obj \), or image distance \( img \) given at least 2 of the other variables. Unlike the matching GUI function imageObject, the units of the 3 variables must be consistent. Note that the image distance \( img \) is equivalent to the principal distance (or camera constant) \( c \).

Example script
imageObject2Example (This script also calls the GUI imageObject function. The examples of this script can be used to experiment with the GUI)

Equations
The Gaussian object-image relationship is given by
\[
\frac{1}{f} = \frac{1}{obj} + \frac{1}{img}
\]
where \( f \) is the focal length, \( obj \) is the object distance, and \( img \) is the image distance (which is equivalent to the camera constant \( c \)).
From the Gaussian object-image relationship any one of the 3 variables can be determined if two of the other variables are known
\[
f = \left( \frac{1}{obj} + \frac{1}{img} \right)^{-1}
\]
\[
obj = \left( \frac{1}{f} - \frac{1}{img} \right)^{-1}
\]
\[
img = \left( \frac{1}{f} - \frac{1}{obj} \right)^{-1}
\]
imageObject

**Purpose**
GUI to solve for focal length, object distance, or image distance, given any 2 of the 3 parameters. Also has plotting option for image distance versus object distance. (imageObject2 is a simplified non-GUI version of this function)

**Syntax**
imageObject

**Arguments**
none

**Output**
output to edit boxes for focal length, object distance, or image distance; plot of image distance versus object distance

**Remarks**
The function imageObject uses the Gaussian object-image relationship to determine any 1 of the 3 variables focal length \( f \), object distance \( obj \), or image distance \( img \) given at least 2 of the other variables. The units of the 3 variables can be mixed by selecting the appropriate radio button for either \( mm \) or \( inch \). The calculation of any of the 3 variables is in the units specified by its units radio button. The plot of image distance versus object distance can also accommodate mixed units as determined by the unit radio buttons for \( obj \) and \( img \). The focal length \( f \) is displayed in the title of the plot in whichever units is was last calculated (or entered). Note that the image distance \( img \) is equivalent to the principal distance (or camera constant) \( c \).

**Example script**
imageObject2Example (Same example script as for the non-GUI imageObject2 function. The examples of this script can be used to experiment with the GUI)

**Required files**
imageObject.fig (GUI figure)

**Equations**
The Gaussian object-image relationship is given by

\[
\frac{1}{f} = \frac{1}{\text{obj}} + \frac{1}{\text{img}}
\]

where \( f \) is the focal length, \( \text{obj} \) is the object distance, and \( \text{img} \) is the image distance (which is equivalent to the camera constant \( c \)).

From the Gaussian object-image relationship any one of the 3 variables can be determined if two of the other variables are known

\[
f = \left( \frac{1}{\text{obj}} + \frac{1}{\text{img}} \right)^{-1}
\]

\[
\text{obj} = \left( \frac{1}{f} - \frac{1}{\text{img}} \right)^{-1}
\]

\[
\text{img} = \left( \frac{1}{f} - \frac{1}{\text{obj}} \right)^{-1}
\]
imagePrelim

Purpose
GUI for preliminary target locations on digital images

Syntax
imagePrelim

Arguments
none

Output
Digital image output to figure with bounding boxes as determined by the regionprops function (image processing toolbox). Automatically generated target IDs (from regionprops) can be overlaid. Binary and grayscale centroid files, as well as manually selected targets, can be saved as text files. Centroid files can be overlaid on the image. An output file consisting of target IDs from one file and centroids from another (within a user specified match tolerance) can be saved.

Remarks
The function imagePrelim is useful for preliminary target locations and analysis of digital images. The GUI should be useful for investigating various strategies for automated target location as well as useful for finding target locations in situations where automation fails. The GUI should be especially useful for images used in camera calibration. The GUI utilizes the regionprops function that operates on binary images. A pushbutton enables selection of the appropriate digital image file (via a popup file selection window) for loading and displaying in a figure window within the GUI. The image is displayed in grayscale, but all preliminary processing is accomplished with a binary version of the image. The initial threshold for the binarization when the image file is first imported is determined by the graythresh function (utilizing Otsu’s method) from the image processing toolbox. A label image is then created from the binary image using bwlabel. The regionprops function is then used to create a structure containing the binary centroids and bounding boxes of each labeled region within the label image. The bounding boxes for each potential target (some of which may potentially be false targets) are overlaid on the image. A larger cross is plotted for very small (and usually false) targets smaller than 3 pixels to improve their identification. The number of targets found, as well as the relative threshold (ranging from 0 to 1), are displayed in text boxes. A slider box (with display) can then be used to interactively adjust the threshold. The newly found targets based on the just selected threshold are overlaid on the image so that one can interactively quickly determine a suitable threshold to automatically find all the valid targets. Typically the highest threshold that finds all the valid targets is selected before possible further processing with the GUI (if additional false targets are found). Slider bars for minimum and maximum bounding box size can then be used to interactively limit the targets found. Selection of a new image or threshold for the current image reinitializes the process. A pushbutton can be used to invert the grayscale before inputting a digital image file (via a popup file selection panel) for cases with black targets on a white background instead of the default white on black. The file name of the inputted digital image is displayed on the GUI along with the number of targets found. The global threshold found from the graythresh function is very appropriate for high contrast targets, but may not work for relatively low contrast targets with a cluttered background. For those cases a pushbutton is available to view the binary image (without further processing) instead of the grayscale image as the threshold is changed via a slider bar. The user can then pick a threshold that best discriminates the targets of interest. Once a suitable threshold has been automatically generated or selected, the user can examine an overlay of bounding boxes around each target to determine minimum and maximum bounding box limits for target selection. The target ID numbers and preliminary binary centroid data can be saved in text format (with user selected file name via file dialog box) with point number, x and y centroid data, half-width, and half-
height of each bounding box respectively. This capability is useful in addition when the binary file is used as input (start values) for full grayscale centroiding. A toggle button to show the binary image without processing aids in preliminary analysis of cluttered images which can be very time consuming when the `regionprops` processing is undertaken at each change of the grayscale threshold. Thus an appropriate threshold can be determined by examination of the binary image before initiating the processing via the `regionprops` function. In this mode all processing except for the slider threshold is disabled until the get image file pushbutton is activated to restart the process. An additional pushbutton allows for manual selection (via mouse) of target ID numbers and the subsequent saving of that xpixel and ypixel data along with the corresponding target ID as a text file (with user selected file name via file dialog box). This additional pushbutton should help in cases where the automatically generated centroid data does not have the desired numbering system. A panel allows the selection of a centroid file to be overlaid on the image. For this overlay panel it is assumed that the first three columns of the data from the file are in order target ID, x, and y. The next 2 columns, if they exist, are taken to be the half-height and half-width of the bounding boxes. A text entry box is available to specify a single value for the bounding box width and height (full width) for files of only 3 columns, which is then used in the overlay plot for all targets. Both the bounding boxes and target IDs are plotted in a color chosen from a popup menu of color selections to aid in discrimination of multiple plots overlaid on the same image. Another panel allows 2 centroid files to be combined into a new file, getting the correct target IDs from 1 file and the correct centroid data from another using the `matchIDs` function. The match tolerance (x, y pixel values must be within this set tolerance to match) is set from within an edit box. Another panel added to the image processing GUI allows grayscale centroiding (with automated perimeter background removal) and output to a new file. This panel is convenient for computing grayscale centroids using the binary centroid files created within the GUI itself as start values. The additional width and height to be added to the binary bounding boxes is entered through an edit box. This helps to minimize clipping of the target since grayscale below the threshold (set to zero during the binarization of the image) may be outside the bounding box found from the binary image, but still may be a valid part of the target. Another panel offers the option of taking threshold and size restrictions from the edit boxes corresponding to the sliders. A separate process button within the panel must be pressed to initiate image processing based on the values in the edit boxes. (The sliders for threshold, min size, and max size are ignored if the edit boxes radio button is selected. When the process button is selected, the values for threshold, min size, and max size are then taken from the corresponding edits boxes as entered by the user instead of from the sliders.) This greatly speeds up preliminary investigations with large format images of several megapixel or more compared to slider selection since with the sliders computations are made at intermediate positions as the sliders are moved toward their final destinations. A pushbutton can be used to select a polynomial region of the image (using the `roiPolySelect` function) in order to remove regions of the image that might contain false targets that are especially hard to discriminate with threshold or size limits.

**Example script**  none

**Required files**
- `imagePrelim.fig` (GUI figure)

**IMAGE PROCESSING TOOLBOX:**
- `bwlabel`
- `getimage`
- `graythresh`
- `imcomplement`
- `imshow`
- `im2bw`
- `regionprops`

**PHOTOGRAMMETRY TOOLBOX:**
- `centroid`
findBackground
matchIDs
overlayCentroidsBox
pixelXYselect
roiPolySelect
multi-camera photogrammetric spatial intersection to determine 3D coordinates given camera parameters and image coordinates from 2 or more cameras (or views)

Syntax

\[
[XYZ] = \text{intersection}(\text{cam})
\]

Arguments

cam
structure array with at least the fields as follows, with N being the camera number:

\(\text{cam}(N).c\)
principal distance \(c\) (or camera constant), usually \(\text{mm}\)

\(\text{cam}(N).xp\)
\(x\)-value of the photogrammetric principal point, usually \(\text{mm}\), but always same units as \(c\).

\(\text{cam}(N).yp\)
\(y\)-value of the photogrammetric principal point, usually \(\text{mm}\), but always same units as \(c\).

\(\text{cam}(N).m\)
\(3 \times 3\) rotation matrix, usually from function \text{rotationMatrix}

\(\text{cam}(N).Xc\)
\(X\)-coordinate of camera perspective center, always same units as \(XYZ\) object coordinates

\(\text{cam}(N).Yc\)
\(Y\)-coordinate of camera perspective center, always same units as \(XYZ\) object coordinates

\(\text{cam}(N).Zc\)
\(Z\)-coordinate of camera perspective center, always same units as \(XYZ\) object coordinates

\(\text{cam}(N).xymm\)
\(M \times 3\) numeric array containing \([\text{pntNum} \ x_{\text{mm}} \ y_{\text{mm}}]\) for each image coordinate for each camera (or view) where \(M\) is the number of image coordinates for a particular camera. \(M\) and the actual point numbers used can vary from camera to camera. Results are returned for any point number that is seen by at least 2 cameras.

Output

\(XYZ\)
\(P \times 8\) numeric array, where \(P\) is the number of points that are seen by at least 2 cameras, of the form below (with units same as perspective center location, \(X_c, Y_c, Z_c\)):

\[
\begin{align*}
pt_1 & \begin{bmatrix} X_1 \ Y_1 \ Z_1 \ X_{1\text{std}} \ Y_{1\text{std}} \ Z_{1\text{std}} \ CamNum_1 \end{bmatrix} \\
pt_2 & \begin{bmatrix} X_2 \ Y_2 \ Z_2 \ X_{2\text{std}} \ Y_{2\text{std}} \ Z_{2\text{std}} \ CamNum_2 \end{bmatrix} \\
\cdot & \\
pt_P & \begin{bmatrix} X_P \ Y_P \ Z_P \ X_{P\text{std}} \ Y_{P\text{std}} \ Z_{P\text{std}} \ CamNum_P \end{bmatrix}
\end{align*}
\]
where $\sigma_X$\textsubscript{N}, $\sigma_Y$\textsubscript{N}, and $\sigma_Z$\textsubscript{N} are the standard deviations of the 3 coordinates from the least squares reduction and $\text{CamNum}_N$ is the number of cameras used for each point in the reduction.

**Remarks**

The function `intersection` is a multi-camera photogrammetric spatial intersection to determine 3D coordinates, given camera parameters and image coordinates from 2 or more cameras (or views). Missing or extra target point numbers for any camera are accommodated. There is no practical limit on the number of cameras that can be passed to the function by means of the structure array `cam`.

**Example script**

`intersectionExample.m` with input files ‘Sample Files\plate11.txt’, ‘Sample Files\camdata1.txt’ and ‘Sample Files\camdata2.txt’

**Equations**

The collinearity equations are given by:

\[
\begin{align*}
X &= x_p - c \left[ m_{11} (X - X_p) + m_{12} (Y - Y_p) + m_{13} (Z - Z_p) \right] \\
Y &= y_p - c \left[ m_{21} (X - X_p) + m_{22} (Y - Y_p) + m_{23} (Z - Z_p) \right]
\end{align*}
\]

The collinearity equations above can be recast in the following form

\[
\begin{align*}
a_1 X + a_2 Y + a_3 Z &= a_1 X_c + a_2 Y_c + a_3 Z_c \\
a_4 X + a_5 Y + a_6 Z &= a_4 X_c + a_5 Y_c + a_6 Z_c
\end{align*}
\]

where

\[
\begin{align*}
a_1 &= (x - x_p) m_{31} + cm_{11} \\
a_2 &= (x - x_p) m_{32} + cm_{12} \\
a_3 &= (x - x_p) m_{33} + cm_{13} \\
a_4 &= (y - y_p) m_{31} + cm_{21} \\
a_5 &= (y - y_p) m_{32} + cm_{22} \\
a_6 &= (y - y_p) m_{33} + cm_{23}
\end{align*}
\]

$X, Y, Z$ is found by linear least squares, where there is 1 pair of ‘a’ equations above (associated with the $x$ and $y$ image coordinates) for each camera for each point. A matrix $A$ is formed that is $2 \times \text{CamNum}$ rows by 3 columns and a $B$ matrix is formed that is $2 \times \text{CamNum}$ rows by 1 column. For instance the $A$ and $B$ matrices would be $4 \times 3$ and $4 \times 1$ respectively when 2 cameras view a single point and $8 \times 3$ and $8 \times 1$ for 4 cameras.
where $A \setminus B$ is the MATLAB operator for linear least squares. Estimates of the standard deviation of $X$, $Y$, and $Z$ are found within the least squares reduction as

$$V = [A] \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} - [B]$$

$$S_0 = \sqrt{V^T V}$$

$$\text{cov} = \left[ \left( [A^T][A] \right)^{1/2} \right]$$

$$\begin{bmatrix} X_a \\ Y_a \\ Z_a \end{bmatrix} = S_0 \sqrt{\text{cov}_{\text{diag}}}$$

where $V$ is a column vector of residuals, $S_0$ is the standard deviation of unit weight, cov is the covariance matrix, cov_{diag} represents the diagonal elements of the covariance matrix, and $X_a$, $Y_a$, $Z_a$ are the estimates of the standard deviation of $X$, $Y$, $Z$ from least squares.
**Purpose**
Linear least squares estimation of Euler rotational angles \((\alpha, \beta, \kappa)\) given other parameters

**Syntax**
\[
[dx, xxp] = \text{lleast3}(\text{angles,XYZc,interior,format,xyimagd,xyimagu,xyzobj})
\]

**Arguments**
- **angle**
  1-column array of estimated \((\alpha, \beta, \kappa)\)

- **XYZc**
  1-column array of the estimated camera position:
  \((X_c, Y_c, Z_c)\)

- **interior**
  1-column array of the given interior orientation parameter:
  \((c_1, c_2, y, z, l_1, l_2, K_1, K_2, P_1, P_2)\)

- **format**
  1-column array containing the camera format data:
  - Number of horizontal pixels
  - Number of vertical pixels
  - Horizontal pixel spacing (mm/pixel)
  - Vertical pixel spacing (mm/pixel)

- **xyimagd**
  2-column array of the distorted image coordinates \((x, y)\) of a set of targets in pixels

- **xyimagu**
  2-column array of the undistorted image coordinates \((x, y)\) of a set of targets in pixels

- **xyzobj**
  3-column array of the object space coordinates \((X, Y, Z)\) of a set of targets, and the units are consistent with \((X_c, Y_c, Z_c)\) (typically in inches)

**Output**
- **dx**
  residual of least squares estimation for all the targets in the image plane

- **xxp**
  estimated distorted \(x_p\)

**Remarks**
This function is used in ‘dlt.m’.

**Called by**
resec3.m
Linear least squares estimation of the camera exterior orientation parameters

Syntax

\[ [\text{dx}, \text{xxp}] = \text{lleast} (\text{exterior}, \text{interior}, \text{format}, \text{xyimagd}, \text{xyimagu}, \text{xyzobj}) \]

Arguments

\- **exterior**: 1-column array of the estimated exterior orientation parameter: 
  \( (\omega, \psi, \kappa, X_c, Y_c, Z_c) \)

\- **interior**: 1-column array of the given interior orientation parameter: 
  \( (c, x_p, y_p, S_h / S_v, K_1, K_2, P_1, P_2) \)

\- **format**: 1-column array containing the following camera format data:
  - Number of horizontal pixels
  - Number of vertical pixels
  - Horizontal pixel spacing (mm/pixel)
  - Vertical pixel spacing (mm/pixel)

\- **xyimagd**: 2-column array of the distorted image coordinates \((x, y)\) of a set of targets in pixels

\- **xyimagu**: 2-column array of the undistorted image coordinates \((x, y)\) of a set of targets in pixels

\- **xyzobj**: 3-column array of the object space coordinates \((X, Y, Z)\) of a set of targets, and the units are consistent with \((X_c, Y_c, Z_c)\) (typically in inches)

Output

\- **dx**: residual of least squares estimation for all the targets in the image plane

\- **xxp**: estimated distorted \(x_p\)

Remarks

This function is used for Newton-Raphson iteration in ‘resec.m’.

Called by

resec.m, resecA.m

Equations

The description of this step in the optimization method for camera calibration/orientation is given in the following reference.

Liu, T., Cattafesta, L., Radezsky, R., and Burner, A. W., “Photogrammetry applied to wind tunnel testing”, AIAA J. Vol. 38, No. 6, 2000, pp. 964-971
**loadCamStruct**

**Purpose**
Loads camera parameter structure from a text file usually created with the matching function `saveCamStruct`.

**Syntax**
```
cam = loadCamStruct(fileName)
```

**Arguments**
- **fileName**
  - fileName of file (with path if necessary) from which to load camera parameter structure (such as the string 'fileName' or the string variable `filename`).

  input text file should be in the form of:
  ```
c = 25.00000
xp = 0.50000
yp = -0.50000
m = 0.4924038765061041
m = -0.5868240888334652
m = 0.6427876096865393
m = 0.870019037522058
m = 0.3104684609733676
m = -0.383022215594890
m = 0.0252013862574872
m = 0.7478280708194912
m = 0.6634139481689384
Xc = 10.00000
Yc = 20.00000
Zc = 30.00000
```

**Output**
- **cam**
  - camera parameter structure with fields as follows:

  - **cam.c**
    - principal distance $c$ (or camera constant), usually $mm$

  - **cam.xp**
    - $x$-value of the photogrammetric principal point, usually $mm$, but always same units as $c$.

  - **cam.yp**
    - $y$-value of the photogrammetric principal point, usually $mm$, but always same units as $c$.

  - **cam.m**
    - $3 \times 3$ rotation matrix, usually from function `rotationMatrix`

  - **cam.Xc**
    - $X$-coordinate of camera perspective center, always same units as $XYZ$ object coordinates

  - **cam.Yc**
    - $Y$-coordinate of camera perspective center, always same units as $XYZ$ object coordinates

  - **cam.Zc**
    - $Z$-coordinate of camera perspective center, always same units as $XYZ$ object coordinates
loadCamStruct is a simple function to load the basic camera parameter structure from a text file, usually created with the matching function saveCamStruct. The function loadCamStruct can be used to load the camera parameter structure into a structure variable within a script or function for further application. The rotation matrix $m$ is assumed saved in row order (default for MATLAB) in the order $m_{11}, m_{21}, m_{31}, m_{22}, m_{23}, m_{32}, m_{33}$. Note that this simple function is only designed to work with the fields of the camera parameter structure identified above. The current version of this simple function has minimal error handling.

**Example script**  
loadCamStructExample.m with input file 'Sample Files\cam1.txt'
locating_target1_fun

**Purpose**
Determination of centroid of a single target at the selected row and column in image

**Syntax**
\[
[xc1\_shifted, yc1\_shifted] =
locating\_target1\_fun(I,row\_p,col\_p,bk\_size\_0)
\]

**Arguments**
- \(I\)
  Image intensity field
- \((row\_p, col\_p)\)
  row and column picked for locating a target
- \(bk\_size\_0\)
  block size for initial searching a target (such as 10 pixels)

**Output**
- \(xc1\_shifted, yc1\_shifted\)
  final target centroid in pixels

**Remarks**
It is assumed in this function that targets in image have higher intensity than background. For dark targets on lighter background, image should be inverted before the use of this function.

**Called by**
- clicking_target_fun.m

**Equations**
The target centroid \((x_c, y_c)\) is defined as
\[
\begin{align*}
x_c &= \frac{\sum_{x_i} \sum_{y_i} x_i I(x_i, y_i)}{\sum_{x_i} \sum_{y_i} I(x_i, y_i)} \\
y_c &= \frac{\sum_{x_i} \sum_{y_i} y_i I(x_i, y_i)}{\sum_{x_i} \sum_{y_i} I(x_i, y_i)}
\end{align*}
\]
where \(I(x_i, y_i)\) is the gray level on an image. When a target contains only a few pixels and the target contrast is not high, the centroid calculation using the above definition may not be accurate.
matchIDs

Purpose
matches correct centroids from one array with correct target IDs of another array (with only approximate centroids). Useful for applying the correct target IDs to automatically generated centroid data, given the correct IDs at approximately the same image locations.

Syntax
centMatch = matchIDs(centID, centCentroid, tol)

Arguments
centID
N × 3 array ([pnt xpix ypix] per row) with correct IDs, but only approximate xpix, ypix image locations. Usually manually created via mouse.

\[
\begin{align*}
  pt_1 & : x_1 \ y_2 \\
  pt_2 & : x_2 \ y_2 \\
  \vdots & \vdots \\
  pt_N & : x_N \ y_N \\
\end{align*}
\]

centCentroid
N × 3 array ([pnt xpix ypix] per row) with (possibly) incorrect IDs, but with correct xpix, ypix image location. Usually automatically generated by way of image processing.

tol
tolerance in pixels used for match criteria between centroid doublets in arrays centID and centCentroid.

Output
centMatch
N × 3 array ([pnt xpix ypix] per row) with correct IDs matched to correct xpix, ypix image locations.

Remarks
The function matchIDs is useful for cases in which correct target labels (or IDs) are available at approximately the same locations as automatically generated (and typically more accurate) centroid data. The automatically generated data will typically not have the correct target labels (IDs) needed for further automated image analyses. The two input argument arrays do not need to be the same size and are not limited to 3 column arrays. However the order of the first three columns must be target ID, x, and then y pixel location. If that is not the case, the proper 3 ordered columns should be selected from the appropriate array for use as input argument array. Note that if the match tolerance is less than the absolute difference between centroid doublets then that match is not made. If that occurs for all rows of the input array centCentroid for a particular target ID, then that target ID does not appear in the output array centMatch. It is useful to compare the size (number of rows) of input array centID and output array centMatch to determine if any target IDs are missing from the output array. (For instance, with [size(centID,1) size(centCentMatch,1)].)

Example script
matchIDsExample.m with input files ‘Sample Files\cent.txt’ and ‘Sample Files\cent2.txt’
mm2pixel

Purpose
Convert image coordinates from mm to pixels

Syntax
xypix = mm2pixel(xymm, Sh, Sv, x0, y0)

Arguments
xymm
array of image coordinates (mm) with point numbers (N × 3) or without point numbers (N × 2), where N = number of image points

xymm with point numbers (3 × 3 array):
1.0000  -2.8587   0.5174
2.0000  -0.2548   1.1635
4.0000  -1.5548  -0.7878

xymm without point numbers (3 × 2 array):
-2.8587   0.5174
-0.2548   1.1635
-1.5548  -0.7878

note that for the 2nd example without point numbers the 3rd doublet of x, y values (-1.5548 - 0.7878) would be taken as point number 3 instead of point number 4 as in the 1st example where point numbers are explicitly entered (see output examples below)

Sh
horizontal pixel spacing in mm. ex: 0.013

Sv
vertical pixel spacing in mm. ex: 0.013

x0, y0
location of image reference center, pixels. For example, a 640 × 480 (Horz × Vert) image would normally be referenced to x0, y0 = 320, 240. x0, y0 locates the center (0, 0) of the image coordinates in mm

Output
xypix
output is an N × 3 array with either explicitly entered point numbers or sequential point numbers from 1:N

output for 1st example of xymm input and Sh, Sv, x0, y0 above:
1.0000  100.1000  200.2000
2.0000  300.4000  150.5000
4.0000  200.4000  300.6000

output for 2nd example of xymm input above:
1.0000  100.1000  200.2000
2.0000  300.4000  150.5000
3.0000  200.4000  300.6000

Remarks
In the function mm2pixel it is assumed that the origin of the outputted image coordinates in pixels is located at the usual upper left of the image with the x-coordinate (horizontal) positive
to the right and the y-coordinate (vertical) positive downward. The origin of the inputted
image coordinates in mm is centered at \(x_0, y_0\) with the x-coordinate positive to the right and
the y-coordinate positive upward. It is common to simply take \(\frac{1}{2}\) of the horizontal and
vertical pixel image dimensions as the values to be used for \(x_0, y_0\) even though the half way
point would actually be \(\frac{1}{2}\) the pixel count + 0.5 pixel. Thus for the 640 \(\times\) 480 (Horz \(\times\) Vert)
image example used above, the actual center of the image in pixels is 320.5, 240.5 rather than
320, 240. However, \(x_0, y_0\) is simply a common reference point on the image. For instance if
the values of 320, 240 are used instead of 320.5, 240.5 for \(x_0, y_0\), then the locations of the
photogrammetric principal point or point of symmetry for distortion would adjust to
accommodate the 0.5 pixel apparent discrepancy yielding the same photogrammetric results
in either case.

Example script

mm2pixelExample.m with input files ‘Sample Image Coordinates\mm2.txt’ and
‘Sample Images\image2.tif’

Equations

\[
x'_{\text{pix}} = x'_o + \frac{x_{\text{mm}}}{S_h}
\]

\[
y'_{\text{pix}} = y'_o - \frac{y_{\text{mm}}}{S_v}
\]
Purpose
Overlays box on current image centered on centroids

Syntax
overlayCentroidsBox(fileName, delx, dely, plotColor)

Arguments
fileName
text file name of \(N \times 2\) (without target numbers, \(x\)-value and \(y\)-value in 1st and 2nd columns respectively) or \(N \times 3\) (with target numbers in 1st column, \(x\)-value and \(y\)-value in 2nd and 3rd columns respectively) saved array of pixel \(x\), \(y\) values; or the array variable itself \((N \times 2\) or \(N \times 3\)); \(N\) is number of centroid \(x\)-\(y\) pairs; examples of \(xypix\) with \(N = 3\) follow:

\[
xypix\text{ with point numbers } (3 \times 3 \text{ array}):
1 \quad 100.1 \quad 200.2 \\
2 \quad 300.4 \quad 150.5 \\
4 \quad 200.4 \quad 300.6 
\]

\[
xypix\text{ without point numbers } (3 \times 2 \text{ array}):
100.1 \quad 200.2 \\
300.4 \quad 150.5 \\
200.4 \quad 300.6 
\]

Note that for the 2nd example without point numbers the 3rd doublet of \(x\), \(y\) values \((200.4 \ 300.6)\) would be taken as point number 3 instead of point number 4 as in the 1st example where point numbers are explicitly entered.

delix
half-width of box centered on each centroid; same for all targets if scalar; unique value for each target if entered as a vector with the same number of elements as targets.

dely
half-height of box centered on each centroid; same for all targets if scalar; unique value for each target if entered as a vector with the same number of elements as targets.

plotColor
optional 4th input string argument to specify the color of the overlay plots. Valid entries are ‘r’ (default), ‘b’, ‘g’, ‘c’, ‘m’, ‘y’, or ‘k’ which indicate respectively red, blue, green, cyan, magenta, yellow, or black.

Output
overlay of boxes of half-width delx and half-height dely on current image centered on centroids

Remarks
In the function overlayCentroidsBox it is assumed that the origin of the image coordinates in pixels is located at the usual upper left of the image with the \(x\)-coordinate (horizontal) positive to the right and the \(y\)-coordinate (vertical) positive downward. The upper left pixel has coordinates of \((1, 1)\).

Example script
overlayCentroidsBoxExample.m with input files 'Sample Files\centroids2.txt' and 'Sample Files\image2.tif'
Purpose

Convert image coordinates from pixels to mm

Syntax

\[ \text{xymm} = \text{pixel2mm}(\text{xypix}, \text{Sh}, \text{Sv}, \text{x0}, \text{y0}) \]

Arguments

- \( \text{xypix} \): array of centroids (pixels) with point numbers (\( N \times 3 \)) or without point numbers (\( N \times 2 \)), where \( N = \) number of image points
  - with point numbers (\( 3 \times 3 \) array):
    - 1 100.1 200.2
    - 2 300.4 150.5
    - 4 200.4 300.6
  - without point numbers (\( 3 \times 2 \) array):
    - 100.1 200.2
    - 300.4 150.5
    - 200.4 300.6

- \( \text{Sh} \): horizontal pixel spacing in mm. ex: 0.013
- \( \text{Sv} \): vertical pixel spacing in mm. ex: 0.013
- \( \text{x0}, \text{y0} \): location of image reference center, pixels. For example, a 640 × 480 (Horz × Vert) image would normally be referenced to \( \text{x0}, \text{y0} = 320, 240 \). \( \text{x0}, \text{y0} \) locates the center \((0, 0)\) of the image coordinates in mm

Output

- \( \text{xymm} \): output is an \( N \times 3 \) array with either explicitly entered point numbers or sequential point numbers from 1:N

If \( \text{Sh} = \text{Sv} = 0.013; \text{x0} = 320; \text{y0} = 240 \) then

- output for 1\textsuperscript{st} example of \( \text{xypix} \) input above:
  - 1.0000 -2.8587 0.5174
  - 2.0000 -0.2548 1.1635
  - 4.0000 -1.5548 -0.7878

- output for 2\textsuperscript{nd} example of \( \text{xypix} \) input above:
  - 1.0000 -2.8587 0.5174
  - 2.0000 -0.2548 1.1635
  - 3.0000 -1.5548 -0.7878
Remarks

In the function `pixel2mm` it is assumed that the origin of the image coordinates in pixels is located at the usual upper left of the image with the x-coordinate (horizontal) positive to the right and the y-coordinate (vertical) positive downward. The origin of the outputted image coordinates in mm is centered at \( x_0, y_0 \) with the x-coordinate positive to the right and the y-coordinate positive upward. It is common to simply take \( \frac{1}{2} \) of the horizontal and vertical pixel image dimensions as the values to be used for \( x_0, y_0 \) even though the half way point would actually be \( \frac{1}{2} \) the pixel count + 0.5 pixel. Thus for the 640 × 480 (Horz × Vert) image example used above, the actual geometrical center of the image in pixels is 320.5, 240.5 rather than 320, 240. However, \( x_0, y_0 \) is simply a common reference point on the image. For instance, if the values of 320, 240 are used instead of 320.5, 240.5 for \( x_0, y_0 \), then the locations of the photogrammetric principal point or point of symmetry for distortion would adjust to accommodate the 0.5 pixel discrepancy in reference point, yielding the same photogrammetric results in either case.

Example script

`pixel2mmExample.m` with input files 'Sample Image Coordinates\centroids2.txt' and 'Sample Images\image2.tif'

Equations

\[
\begin{align*}
x_{\text{mm}} &= (x_{\text{pix}} - x_0)S_h \\
y_{\text{mm}} &= -(y_{\text{pix}} - y_0)S_v
\end{align*}
\]
Purpose

manual selection of image coordinates with mouse and storage to file

Syntax

pixelXYselect
XY = pixelXYselect
XY = pixelXYselect(i)
XY = pixelXYselect('FileName', 'ffff', 'Nstart', i, 'FileMode', 'm', 'fig', g, 'PrintOut', p)

In the 1st simplest calling syntax above the target location selections are made on the current, or active figure (by invoking gcf). Selected locations [target #, x, y] in pixel output are appended to the default file 'centTemp.txt' in the current directory with a starting number of 1.

The 2nd syntax, in addition, puts the [targ#, x, y] locations in variable XY.

The 3rd syntax, perhaps the most friendly syntax due to its simplicity and usefulness, is used to set the starting target number. With the 3rd syntax large missing sections of target numbers can be handled by recalling the function with the next target number in the sequence taken to be the new starting number. Once all the targets have been selected, the default file centTemp.txt can be renamed and edited. It is suggested that for missing targets that do not span a wide range, that the cursor be placed to the left of the image to yield negative x-values, which can be readily picked out and removed during editing (or a script can be written to throw out negative values automatically).

The 4th syntax is the most general and must be used for changing default values other than starting target number. The new values must be entered as argument pairs where the 1st string of the pair specifies the argument label and the 2nd entry (a character string for file name and mode, a numeric value for starting number and figure number) specifies the value of the argument used in the function. Argument labels specified this way are file name 'FileName', file mode 'FileMode', figure number 'fig', and when other arguments are specified, the starting target number 'Nstart' must then be also specified in an argument pair. The argument labels must match exactly the above entries including case. In the 4th syntax, 'ffff' represents the character string (or string variable) for the file name, i represents the starting numerical value (or numeric variable) of starting target number, 'm' represents the file mode which can be generally either 'a' for append or 'w' for write (without appending), 'g' represents the numerical value of the figure number to be used, and 'p' represents either 0 (no printout to the command window) or 1 (printout).

Arguments

i
starting target number, which can be a single numerical value (or variable) argument. Default is 1

'Nstart', i
if other arguments in addition to starting target number are entered, then the starting target number must also be entered paired with its argument label 'Nstart'

'FileName', 'ffff'
'FileName' is the argument label which must be paired with the file name character string (or string variable) 'ffff'. Default is 'centTemp.txt'
'FileMode', 'm'
'FileMode' is the argument label which must be paired with the file name character string (or string variable) 'm', which normally would either be 'a' for append or 'w' for write (without appending). Default is 'a'

'fig', g
'fig' is the argument label which must be paired with the numerical value (or numeric variable) g. Default is current figure (by means of gcf)

'PrintOut', p
'Printout' is the argument label which must be paired with off (0) or on (1) for printout to the command window. Default is 1.

Output
XY output is an N × 3 array with sequential target numbers from Nstart:[Nstart + Ntargs – 1]
where Ntargs represents the number of selected target locations

example output
1   405   404
2   459   373
3   466   308

Example script
pixelXYselectExample.m with input files 'Sample Files\image1.tif' and 'Sample Files\image2.tif'. Output is written to centTemp.txt and centTemp2.txt in the current MATLAB directory.

Remarks
Use img = imshow('imageFileName') to put the image for the file 'imageFileName' in a figure before calling function pixelXYselect. Either select the figure containing the image or use the MATLAB figure(n) to select the desired figure n (or optionally by an input argument to the function). A useful aid is to invoke iptsetpref('ImshowAxesVisible', 'on') in order to show the pixel axes on the figure when using imshow. Invoke iptsetpref('ImshowAxesVisible', 'off') to reset the imshow option to not show the axes.
PM2Australis

**Purpose**
Convert from PhotoModeler camera orientation angles $\omega, \phi, \kappa$
to Australis camera orientation angles, Azimuth, Elevation, Roll

**Syntax**
AzimuthElevationRoll = PM2Australis(Omega, Phi, Kappa)

**Arguments**
Omega
angle about X-axis, taken as + for CCW rotation; in degrees

Phi
angle about Y-axis taken as + for CCW rotation; in degrees

Kappa
angle about Z-axis taken as + for CCW rotation; in degrees

**Output**
AzimuthElevationRoll
output is a 1 × 3 array of angles in the order Azimuth, Elevation, Roll

**Remarks**
Order of application of angles on input is $\omega, \phi, \kappa$. On output order is Azimuth, Elevation, Roll.

**Example script**
PM2AustralisExample

**Equations**

\[
\begin{align*}
m_{11} &= \cos \phi \cos \kappa \\
m_{12} &= \sin \omega \sin \phi \cos \kappa + \cos \omega \sin \kappa \\
m_{13} &= -\cos \omega \sin \phi \cos \kappa + \sin \omega \sin \kappa \\
m_{21} &= -\cos \phi \sin \kappa \\
m_{22} &= -\sin \omega \sin \phi \sin \kappa + \cos \omega \cos \kappa \\
m_{23} &= \cos \omega \sin \phi \sin \kappa + \sin \omega \cos \kappa \\
m_{31} &= \sin \phi \\
m_{32} &= -\sin \omega \cos \phi \\
m_{33} &= \cos \omega \cos \phi
\end{align*}
\]
where \( \alpha \) = azimuth, \( \varepsilon \) = elevation, and \( \rho \) = roll and \( \omega \), \( \phi \), \( \kappa \) equal the Euler angles omega, phi, kappa. Note that the 4-quadrant inverse tangent function \( \text{atan2}(y, x) \) is used instead of the 2-quadrant \( \text{atan}(y/x) \) (which would have limited computed angles to \( \pm 90^\circ \) instead of \( \pm 180^\circ \)) for the arctangent computations within the function.
Purpose
Determination of the camera responsive function based on the Chebysev functions

Syntax
[coef,residual]=RadiomCali_cheby_fun(R12,zeta1,zeta2,NoTerm)

Arguments
R12
approximate value of $R_{12}$ is given by (see Remarks)
$$R_{12} \approx \frac{m(I_{\text{max},2}) (t_{\text{INT}} / F^2)}{m(I_{\text{max},1}) (t_{\text{INT}} / F^2)}.$$

zeta1
zeta1 is the normalized image intensity of image 1, where $\xi(x) = m[I(x)] / m(I_{\text{max}})$ is the non-dimensional measurement of $I(x)$ normalized by the maximum value and $I_{\text{max}}$ corresponds to the maximum radiance in the scene.

zeta2
zeta2 is the normalized image intensity of image 2, where $\xi(x) = m[I(x)] / m(I_{\text{max}})$ is the non-dimensional measurement of $I(x)$ normalized by the maximum value and $I_{\text{max}}$ corresponds to the maximum radiance in the scene.

NoTerm
The number of the Chebysev functions for the camera responsive function

Output
coef
the coefficients of a set of the Chebysev functions
$$(1, x, 2x^2 - 1, 4x^3 - 3x, 8x^4 - 8x^2 + 1, 16x^5 - 20x^3 + 5x)$$

residual
residual of least squares estimation

Example script
RadiomCali_chebyExample.m

Equations
Radiometric measurements using a CCD camera require a good linear response of the electrical output to the scene radiance. However, there are many stages of image acquisition that may introduce non-linearity; for example, video cameras often include some form of “gamma” mapping. When the radiometric response function of a camera is known, the non-linearity can be corrected. Here, a simple algorithm is described to determine the radiometric response function of a camera from a scene image taken in different exposures. First, we define $I(x)$ as a linear radiometric response to the scene radiance and $m[I(x)]$ as the measurement of $I(x)$ by camera electronic circuitry that may produce a non-linear electrical output. Actually, the measurement $m[I(x)]$ is the brightness or gray level of an image, where $x$ is the image coordinates. The non-dimensional response function relating $I(x)$ to $m[I(x)]$ is defined by
$$\frac{I(x)}{I_{\text{max}}} = f[\xi(x)],$$  \hspace{1cm} (1)
where \( \xi(x) = m[I(x)]/m(I_{\text{max}}) \) is the non-dimensional measurement of \( I(x) \) normalized by the maximum value and \( I_{\text{max}} \) corresponds to the maximum radiance in the scene. Recovery of \( f(\xi) \) is the task of the radiometric calibration of a camera.

Two images of a scene are taken in two different exposures. According to the camera formula, \( I(x) \) is proportional to the integration time \( t_{\text{INT}} \) and inversely proportional to the square of the \( f \)-number \( F \). Thus, we have the following functional equation for \( f(\xi) \),

\[
f(\xi_1)/f(\xi_2) = R_{12},
\]

where the subscripts 1 and 2 denote the image 1 and image 2, and the factor \( R_{12} \) is defined as

\[
R_{12} = \frac{I_{\text{max}2}}{I_{\text{max}1}} \frac{(t_{\text{INT}2}/F^2)}{(t_{\text{INT}1}/F^2)_2}.
\]

Since \( m(I_{\text{max}}) \) corresponds to \( I_{\text{max}} \), the boundary condition for \( f(\xi) \) is \( f(\xi = 1) = 1 \). We assume that \( f(\xi) \) can be expanded as

\[
f(\xi) = \sum_{n=0}^{N} c_n \phi_n(\xi),
\]

where the base functions \( \phi_n(\xi) \) are the Chebyshev functions although other orthogonal functions and non-orthogonal functions like polynomials can also be used. Substitution of Eq. (4) to Eq. (2) leads to the following equations for the coefficients \( c_n \)

\[
\sum_{n=0}^{N} c_n \phi_n(\xi_1) - R_{12} \phi_n(\xi_2) = 0,
\]

\[
\sum_{n=0}^{N} c_n \phi_n(1) = 1.
\]

For selected \( M \) pixels in a scene image, Eq. (5) constitutes a system of \( M+1 \) equations for the \( N+1 \) unknowns \( c_n \) (\( M \geq N \)). For a given \( R_{12} \), a least-squares solution for \( c_n \) can be found. In practice, since the factor \( R_{12} \) is not exactly known a priori, we use an approximate value of \( R_{12} \)

\[
R_{12} \approx \frac{m(I_{\text{max}2})}{m(I_{\text{max}1})} \frac{(t_{\text{INT}2}/F^2)}{(t_{\text{INT}1}/F^2)_2}.
\]

An iteration scheme can be used to give an improved value of \( R_{12} \).

RadiomCali_poly_fun

Purpose
Determination of the camera responsive function based on the power functions

Syntax
[coef,residual]=RadiomCali_poly_fun(R12,zeta1,zeta2,NoTerm)

Arguments
R12
approximate value of $R_{12}$ is given by (see Remarks)

$$R_{12} \approx \frac{m(I_{\text{max},2}) (t_{\text{INT},2} / F^2)}{m(I_{\text{max},1}) (t_{\text{INT},1} / F^2)}.$$

\[ \text{zeta1} \]
\[ \text{zeta1 is the normalized image intensity of image 1, where } \xi(x) = m[I(x)] / m(I_{\text{max}}) \text{ is the non-dimensional measurement of } I(x) \text{ normalized by the maximum value and } I_{\text{max}} \text{ corresponds to the maximum radiance in the scene.} \]

\[ \text{zeta2} \]
\[ \text{zeta2 is the normalized image intensity of image 2, where } \xi(x) = m[I(x)] / m(I_{\text{max}}) \text{ is the non-dimensional measurement of } I(x) \text{ normalized by the maximum value and } I_{\text{max}} \text{ corresponds to the maximum radiance in the scene.} \]

\[ \text{NoTerm} \]
The number of the power functions for the camera responsive function

Output
\[ \text{coef} \]
the coefficients of a set of the power functions
\( (I, x, x^2, x^3, x^4, x^5) \)

\[ \text{residual} \]
residual of least squares estimation

Example script
RadiomCali_polyExample.m

Equations
Radiometric measurements using a CCD camera require a good linear response of the electrical output to the scene radiance. However, there are many stages of image acquisition that may introduce non-linearity; for example, video cameras often include some form of ‘gamma’ mapping. When the radiometric response function of a camera is known, the non-linearity can be corrected. Here, a simple algorithm is described to determine the radiometric response function of a camera from a scene image taken in different exposures. First, we define $I(x)$ as a linear radiometric response to the scene radiance and $m[I(x)]$ as the measurement of $I(x)$ by camera electronic circuitry that may produce a non-linear electrical output. Actually, the measurement $m[I(x)]$ is the brightness or gray level of an image, where $x$ is the image coordinates. The non-dimensional response function relating $I(x)$ to $m[I(x)]$ is defined by

$$I(x) / I_{\text{max}} = f[\xi(x)],$$

(1)
where $\xi(x) = m[I(x)] / m(I_{\text{max}})$ is the non-dimensional measurement of $I(x)$ normalized by the maximum value and $I_{\text{max}}$ corresponds to the maximum radiance in the scene. Recovery of $f(\xi)$ is the task of the radiometric calibration of a camera.

Two images of a scene are taken in two different exposures. According to the camera formula, $I(x)$ is proportional to the integration time $t_{\text{INT}}$ and inversely proportional to the square of the $f$-number $F$. Thus, we have the following functional equation for $f(\xi)$,

$$f(\xi_1) / f(\xi_2) = R_{12}, \quad (2)$$

where the subscripts 1 and 2 denote the image 1 and image 2, and the factor $R_{12}$ is defined as

$$R_{12} = \frac{I_{\text{max},2}}{I_{\text{max},1}} \left( \frac{t_{\text{INT}} / F^2}{t_{\text{INT}} / F^2} \right). \quad (3)$$

Since $m(I_{\text{max}})$ corresponds to $I_{\text{max}}$, the boundary condition for $f(\xi)$ is $f(\xi = 1) = 1$. We assume that $f(\xi)$ can be expanded as

$$f(\xi) = \sum_{n=0}^{N} c_n \phi_n(\xi), \quad (4)$$

where the base functions $\phi_n(\xi)$ are the Chebyshev functions although other orthogonal functions and non-orthogonal functions like polynomials can also be used. Substitution of Eq. (4) to Eq. (2) leads to the following equations for the coefficients $c_n$,

$$\sum_{n=0}^{N} c_n I_{\text{max},1} \phi_n(\xi_1) - R_{12} \phi_n(\xi_2)] = 0, \quad (5)$$

$$\sum_{n=0}^{N} c_n \phi_n(1) = 1. \quad (6)$$

For selected $M$ pixels in a scene image, Eq. (5) constitutes a system of $M+1$ equations for the $N+1$ unknowns $c_n$ ($M \geq N$). For a given $R_{12}$, a least-squares solution for $c_n$ can be found. In practice, since the factor $R_{12}$ is not exactly known a priori, we use an approximate value of $R_{12}$

$$R_{12} \approx \frac{m(I_{\text{max},2}) (t_{\text{INT}} / F^2)}{m(I_{\text{max},1}) (t_{\text{INT}} / F^2)}. \quad (7)$$

An iteration scheme can be used to give an improved value of $R_{12}$.

### Purpose
Determination of Euler rotational angles when other parameters are given

### Syntax

```
[dxp,exterior]=resec3(epsilon,interior,exterior,format,xyimagd,xyimagu,xyzobj)
```

### Arguments

- **epsilon**
  small number for controlling iteration

- **interior**
  1-column array of the interior orientation parameters,
  \((c, x_p, y_p, S_h / S_v, K_x, K_y, P_1, P_2, )\)

- **exterior**
  1-column array of the estimated exterior orientation parameters,
  \((\omega, \varphi, \kappa, X_\epsilon, Y_\epsilon, Z_\epsilon)\)

- **format**
  1-column array containing the following camera format data:
  
  Number of horizontal pixels  
  Number of vertical pixels  
  Horizontal pixel spacing (mm/pixel)  
  Vertical pixel spacing (mm/pixel)

- **xyimagd**
  2-column array of the distorted image coordinates \((x, y)\) of a set of targets in pixels

- **xyimagu**
  2-column array of the undistorted image coordinates \((x, y)\) of a set of targets in pixels

- **xyzobj**
  3-column array of the object space coordinates \((X, Y, Z)\) of a set of targets, and the units are consistent with \((X_\epsilon, Y_\epsilon, Z_\epsilon)\) (typically in inches)

### Output

- **dxp**
  standard deviation of calculated \(x_p\) over all the targets

- **exterior**
  1-column array of the refined exterior orientation parameters,
  \((\omega, \varphi, \kappa, X_\epsilon, Y_\epsilon, Z_\epsilon)\)

### Called by

`dlt.m`
### Purpose
Determination of the exterior orientation parameters (resection) using Newton-Raphson method

### Syntax
`[dxp]=resec(interior,exterior,format,xyimagd,xyimagu,xyzobj,corrindex)`

### Arguments
- **interior**: 1-column array of the interior orientation parameters,
  \( (c, x_p, y_p, S_h / S_v, K_1, K_2, P_1, P_2) \)
- **exterior**: 1-column array of the exterior orientation parameters,
  \( (\omega, \varphi, \kappa, X_c, Y_c, Z_c) \)
- **format**: 1-column array containing the following camera format data:
  - Number of horizontal pixels
  - Number of vertical pixels
  - Horizontal pixel spacing (mm/pixel)
  - Vertical pixel spacing (mm/pixel)
- **xyimagd**: 2-column array of the distorted image coordinates \((x, y)\) of a set of targets in pixels
- **xyimagu**: 2-column array of the undistorted image coordinates \((x, y)\) of a set of targets in pixels
- **xyzobj**: 3-column array of the object space coordinates \((X, Y, Z)\) of a set of targets, and the units are consistent with \((X_c, Y_c, Z_c)\) (typically in inches)
- **corrindex**: The iteration number for lens distortion correction

### Output
- **dxp**: standard deviation of calculated \(x_p\) over all the targets

### Remarks
This function provides an objective function ‘dxp’ for minimization to determine the correct interior orientation parameters.

### Called by
`camcal_fun.m`

### Equations
The detailed description of the optimization method for camera calibration/orientation is given in the following reference.
Liu, T., Cattafesta, L., Radezsky, R., and Burner, A. W., “Photogrammetry applied to wind tunnel testing”, AIAA J. Vol. 38, No. 6, 2000, pp. 964-971
Purpose

Determination of the exterior orientation parameters using the closed-form resection method developed by Zeng and Wang based on three known targets.

Syntax

```
[exterior]=resec_ZW(xyimag,xyzobj,camformat,c)
```

Arguments

- `camformat`: 1-column array containing the following camera format data:
  - Number of horizontal pixels
  - Number of vertical pixels
  - Horizontal pixel spacing (mm/pixel)
  - Vertical pixel spacing (mm/pixel)

- `xyimag`: 2-column array of the image coordinates (x, y) of three targets in pixels

- `xyzobj`: 3-column array of the object space coordinates (X, Y, Z) of three targets, and the units are consistent with (X_e, Y_e, Z_e) in inches

- `c`: the principal distance in mm (approximately focal length)

Output

- `exterior`: two sets of the exterior orientation parameters (ω, φ, κ, X_e, Y_e, Z_e)

Remarks

This closed-form resection function typically gives two sets (two solutions) of the exterior orientation parameters. To determine the correct set, additional information is needed. For example, when an additional known target is given, we have two groups of three known targets. Then, we run 'resec_ZW.m' for the two groups and obtain four sets of the exterior orientation parameters. If one set of the exterior orientation parameters is repeated in two runs, it is the correct one that should remain invariant for different groups of targets. Another important point in the use of this function is that three targets should be numbered in a counterclockwise fashion in both the image plane and object space. This facilitates the selection of the appropriate sets of (X_e, Y_e, Z_e).

Example script

`resec_ZWExample.m`

Equations

The detailed description of the closed-form resection method for the exterior orientation parameters is given in the following reference.

resecA

Purpose
Determination of Euler rotational angles when other parameters are given

Syntax
[dxp,exterior]=resecA(interior,exterior,format,xyimagd,xyimagu,xyzobj)

Arguments
interior
1-column array of the interior orientation parameters,
\((c, x_p, y_p, S_h / S_v, K_1, K_2, P_1, P_2, )\)

exterior
1-column array of the estimated exterior orientation parameters,
\((\omega, \varphi, \kappa, X_c, Y_c, Z_c)\)

format
1-column array containing the following camera format data:
Number of horizontal pixels
Number of vertical pixels
Horizontal pixel spacing (mm/pixel)
Vertical pixel spacing (mm/pixel)

xyimagd
2-column array of the distorted image coordinates \((x, y)\) of a set targets in pixels

xyimagu
2-column array of the undistorted image coordinates \((x, y)\) of a set of targets in pixels

xyzobj
3-column array of the object space coordinates \((X, Y, Z)\) of a set of targets, and the units are consistent with \((X_c, Y_c, Z_c)\) (typically in inches)

Output
dxp
standard deviation of calculated \(x_p\) over all the targets

exterior
1-column array of the refined exterior orientation parameters,
\((\omega, \varphi, \kappa, X_c, Y_c, Z_c)\)

Remarks
This function is used in ‘camcal_fun.m’.

Example script
camcal_fun.m
Resection

**Purpose**

nonlinear least squares (NLLS) to determine $\omega$, $\phi$, $\kappa$, $X_c$, $Y_c$, and $Z_c$ and estimates of the standard deviations of these parameters given camera interior parameters ($c$, $x_p$, $y_p$), image data, and $X$, $Y$, $Z$ object space data.

**Syntax**

camOut = resection(camIn, XYZ)

**Arguments**

- **camIn**
  structure with at least the following fields:
    - **camIn.c**
      principal distance $c$ (or camera constant), usually *mm*
    - **camIn.xp**
      $x$-value of the photogrammetric principal point, usually *mm*, but always same units as $c$.
    - **camIn.yp**
      $y$-value of the photogrammetric principal point, usually *mm*, but always same units as $c$.
    - **camIn.omega**
      angle in degrees about $X$-axis, $\omega$ taken as + for CCW rotation when viewing down the axis toward the origin.
    - **camIn.phi**
      angle in degrees about $Y$-axis, $\phi$ taken as + for CCW rotation when viewing down the axis toward the origin.
    - **camIn.kappa**
      angle in degrees about $Z$-axis, $\kappa$ taken as + for CCW rotation when viewing down the axis toward the origin.
    - **camIn.Xc**
      $X$-coordinate of camera perspective center, always same units as $XYZ$ object coordinates
    - **camIn.Yc**
      $Y$-coordinate of camera perspective center, always same units as $XYZ$ object coordinates
    - **camIn.Zc**
      $Z$-coordinate of camera perspective center, always same units as $XYZ$ object coordinates
    - **camIn.xymm**
      N X 3 numeric array containing [pntNum  xmm  ymm] for each image coordinate seen by the camera

- **XYZ**
  N × 4 numeric array of the form below (with units same as perspective center location, $X_c$, $Y_c$, $Z_c$):

  $$
  pt_1 \quad X_1 \quad Y_1 \quad Z_1 \\
  pt_2 \quad X_2 \quad Y_2 \quad Z_2
  $$
Output

```
camOut
structure with fields as follows:

camOut.c
principal distance $c$ (or camera constant), usually $mm$, echoed from input structure camIn

camOut.xp
$x$-value of the photogrammetric principal point, usually $mm$, but always same units as $c$,
echoed from input structure camIn

camOut.yp
$y$-value of the photogrammetric principal point, usually $mm$, but always same units as $c$,
echoed from input structure camIn

camOut.omega
angle in degrees about $X$-axis, $\omega$ taken as $+$ for CCW rotation when viewing down the axis
toward the origin

camOut.phi
angle in degrees about $Y$-axis, $\phi$ taken as $+$ for CCW rotation when viewing down the axis
toward the origin

camOut.kappa
angle in degrees about $Z$-axis, $\kappa$ taken as $+$ for CCW rotation when viewing down the axis
toward the origin

camOut.Xc
$X$-coordinate of camera perspective center, always same units as $XYZ$ object coordinates

camOut.Yc
$Y$-coordinate of camera perspective center, always same units as $XYZ$ object coordinates

camOut.Zc
$Z$-coordinate of camera perspective center, always same units as $XYZ$ object coordinates

camOut.omegastd
estimated standard deviation of $\omega$ from NLLS, in degrees

camOut.phistd
estimated standard deviation of $\phi$ from NLLS, in degrees

camOut.kappastd
estimated standard deviation of $\kappa$ from NLLS, in degrees

camOut.Xcstd
estimated standard deviation of $X_c$ from NLLS, in degrees

camOut.Ycstd
estimated standard deviation of $Y_c$ from NLLS, in degrees
```
camOut.Zcstd
estimated standard deviation of $Z_c$ from NLLS, in degrees

camOut.So
standard deviation of unit weight from NLLS

camOut.xstd
standard deviation of the $x$-coordinates of the differences between the input image coordinates and the computed coordinates based on resection output parameters

camOut.ystd
standard deviation of the $y$-coordinates of the differences between the input image coordinates and the computed coordinates based on resection output parameters

camOut.xymm
$N \times 3$ numeric array containing $[\text{pntNum} \ x_{\text{mm}} \ y_{\text{mm}}]$ for each image coordinate seen by the camera, echoed from input structure camIn.xymm

Reference
Elements of Photogrammetry, Paul R. Wolf, 2nd edition, McGraw-Hill, p. 606-609, but with the opposite sign for coefficients $b_{11}$ - $b_{13}$ and $b_{21}$ - $b_{23}$, and replacing the symbol for the camera constant $f$ with $c$.

Remarks
Nonlinear least squares (NLLS) is used to determine the exterior orientation parameters $\alpha$, $\phi$, $\kappa$, $X_c$, $Y_c$, and $Z_c$ and estimates of the standard deviations of these parameters given camera interior parameters ($c, x_p, y_p$), image data, and $X, Y, Z$ object space data. The function resection uses the linearization method (sometimes called the Gauss, Gauss-Newton, or Taylor series method) to solve the nonlinear least squares problem. For this method, the collinearity equations are linearized using Taylor’s theorem. This linearization yields 2 equations (1 each for $x$- and $y$-image coordinate) for each 3D point. These equations contain initial approximations and products of the partial derivatives. Corrections are solved for by linear least squares and applied iteratively to the initial approximations to determine the final values of the parameters. The notation follows Wolf’s 2nd edition of Elements of Photogrammetry, pp. 606-609, but with the opposite sign for coefficients $b_{11}$ - $b_{13}$ and $b_{21}$ - $b_{23}$ and the symbol $f$ replaced with $c$. The final estimates of the parameters are found from the over-determined set of equations representing all the 3D locations with common target point numbers in both the $XYZ$ object and $xymm$ image set (2 equations for each 3D location). Note that the correction terms $d\alpha$, $d\phi$, $d\kappa$, $dX_c$, $dY_c$, $dZ_c$ are solved for at each iteration, and that the parameters $\alpha$, $\phi$, $\kappa$, $X_c$, $Y_c$, $Z_c$ themselves are found by interatively adding the correction terms to the parameter values found after the previous iteration. After several iterations the corrections approach zero and the final iterated solutions for the parameters are determined. To avoid the possibility of an endless loop, the function resection uses a fixed number of 20 iterations before exit from the function (instead of testing for corrections that approach negligibly small values as an exit criterion).

Example script
resectionExample.m with input files ‘Sample Files\XYZ1.txt’ and ‘Sample Files\centroids3.txt’

Equations
\[
x = x_p - c \left[ \frac{m_{11}(X - X_c) + m_{12}(Y - Y_c) + m_{13}(Z - Z_c)}{m_{31}(X - X_c) + m_{32}(Y - Y_c) + m_{33}(Z - Z_c)} \right]
\]
\[
y = y_p - c \left[ \frac{m_{21}(X - X_c) + m_{22}(Y - Y_c) + m_{23}(Z - Z_c)}{m_{31}(X - X_c) + m_{32}(Y - Y_c) + m_{33}(Z - Z_c)} \right]
\]
\[
\begin{bmatrix}
\frac{d\omega}{d\phi} \\
\frac{d\phi}{d\kappa} \\
\frac{d\kappa}{dX_c} \\
\frac{dX_c}{dY_c} \\
\frac{dY_c}{dZ_c}
\end{bmatrix} = A\backslash L
\]

where \( A\backslash L \) is the MATLAB operator for linear least squares and the \( A \) and \( L \) matrices are as follows, with \( x, y \) being the image coordinates, \( x_p, y_p \) being the location of the photogrammetric principal point, and \( c \) is the camera constant (principal distance):

\[
A = \begin{bmatrix}
    b_{11} & b_{12} & b_{13} & b_{14} & b_{15} & b_{16} \\
    b_{21} & b_{22} & b_{23} & b_{24} & b_{25} & b_{26} \\
    \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
    \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\end{bmatrix}
\]

\[
L = \begin{bmatrix}
    x - x_p + \frac{rc}{q} \\
    y - y_p + \frac{sc}{q} \\
    \vdots \\
    \vdots \\
\end{bmatrix}
\]

where with

\[
\Delta X = X - X_c \\
\Delta Y = Y - Y_c \\
\Delta Z = Z - Z_c
\]

we have

\[
q = m_{11}\Delta X + m_{12}\Delta Y + m_{13}\Delta Z \\
r = m_{21}\Delta X + m_{22}\Delta Y + m_{23}\Delta Z \\
s = m_{31}\Delta X + m_{32}\Delta Y + m_{33}\Delta Z
\]

and
\[ b_{11} = \frac{x}{q} (m_{13} \Delta Y - m_{31} \Delta Z) + \frac{c}{q} (m_{13} \Delta Y - m_{12} \Delta Z) \]
\[ b_{12} = \frac{x}{q} (-\Delta X \cos \phi - \Delta Y \sin \phi \sin \phi + \Delta Z \cos \phi \sin \phi) \]
\[ + \frac{c}{q} (\Delta X \sin \phi \cos \kappa - \Delta Y \sin \phi \cos \phi \cos \kappa + \Delta Z \cos \phi \cos \phi \cos \kappa) \]
\[ b_{13} = \frac{-c}{q} s \]
\[ b_{14} = \frac{x}{q} m_{31} + \frac{c}{q} m_{11} \]
\[ b_{15} = \frac{x}{q} m_{32} + \frac{c}{q} m_{12} \]
\[ b_{16} = \frac{x}{q} m_{33} + \frac{c}{q} m_{13} \]
\[ b_{21} = \frac{y}{q} (m_{31} \Delta Y - m_{11} \Delta Z) + \frac{c}{q} (m_{31} \Delta Y - m_{21} \Delta Z) \]
\[ b_{22} = \frac{y}{q} (-\Delta X \cos \phi - \Delta Y \sin \phi \sin \phi + \Delta Z \cos \phi \sin \phi) \]
\[ - \frac{c}{q} (\Delta X \sin \phi \sin \kappa + \Delta Y \sin \phi \cos \phi \sin \kappa - \Delta Z \cos \phi \cos \phi \sin \kappa) \]
\[ b_{23} = \frac{-c}{q} r \]
\[ b_{24} = \frac{y}{q} m_{31} + \frac{c}{q} m_{21} \]
\[ b_{25} = \frac{y}{q} m_{32} + \frac{c}{q} m_{22} \]
\[ b_{26} = \frac{y}{q} m_{33} + \frac{c}{q} m_{23} \]

after each iteration the corrections are added to the latest value of the parameters found from the previous iteration

\[ \omega = \omega + d\omega \]
\[ \phi = \phi + d\phi \]
\[ \kappa = \kappa + d\kappa \]
\[ X_c = X_c + dX_c \]
\[ Y_c = Y_c + dY_c \]
\[ Z_c = Z_c + dZ_c \]
Estimates of the standard deviation of $\omega$, $\phi$, $\kappa$, $X_c$, $Y_c$, and $Z_c$ are found within the least squares reduction as

$$V = -L$$

$$S_0 = \sqrt{\frac{V^T [V]}{df}}$$

$$cov = \left([A^T [A]]^{-1} \right)$$

$$\begin{bmatrix}
\omega_{\sigma} \\
\phi_{\sigma} \\
\kappa_{\sigma} \\
X_{c\sigma} \\
Y_{c\sigma} \\
Z_{c\sigma}
\end{bmatrix} = S_0 \sqrt{cov_{diag}}$$

where $V$ is a column vector of residuals, $S_0$ is the standard deviation of unit weight, $df$ is the degrees of freedom, cov is the covariance matrix, $cov_{diag}$ represents the diagonal elements of the covariance matrix, and $\omega_{\sigma}$, $\phi_{\sigma}$, $\kappa_{\sigma}$, $X_{c\sigma}$, $Y_{c\sigma}$, and $Z_{c\sigma}$ are the estimates of the standard deviations of $\omega$, $\phi$, $\kappa$, $X_c$, $Y_c$, and $Z_c$ from least squares.
Purpose

Determines 3 alternate sets of exterior orientation (which are possible local minima) for resection on nearly planar objects. The cal-plate primary lateral dimensions are assumed to be $X, Y$ with $Z \approx$ constant (representing uniform depth).

Syntax

\[
\text{camLocalMin} = \text{resectionLocalMin}(\text{cam}, \text{Zmean})
\]

Arguments

cam
structure with at least the following fields:

cam.omega
angle in degrees about $X$-axis, $\omega$ taken as + for CCW rotation when viewing down the axis toward the origin

cam.phi
angle in degrees about $Y$-axis, $\phi$ taken as + for CCW rotation when viewing down the axis toward the origin

cam.kappa
angle in degrees about $Z$-axis, $\kappa$ taken as + for CCW rotation when viewing down the axis toward the origin

cam.Xc
$X$-coordinate of camera perspective center, always same units as $XYZ$ object coordinates

cam.Yc
$Y$-coordinate of camera perspective center, always same units as $XYZ$ object coordinates

cam.Zc
$Z$-coordinate of camera perspective center, always same units as $XYZ$ object coordinates

Zmean
optional input argument specifying mean of cal-plate $Z$-values if mean $\neq 0$.

Output

camLocalMin
structure with fields as follows:

camLocalMin.omega
angle in degrees about $X$-axis, $\omega$ taken as + for CCW rotation when viewing down the axis toward the origin

camLocalMin.phi
angle in degrees about $Y$-axis, $\phi$ taken as + for CCW rotation when viewing down the axis toward the origin

camLocalMin.kappa
angle in degrees about $Z$-axis, $\kappa$ taken as + for CCW rotation when viewing down the axis toward the origin

camLocalMin.Xc
X-coordinate of camera perspective center, always same units as XYZ object coordinates

\texttt{camLocalMin.Yc}

Y-coordinate of camera perspective center, always same units as XYZ object coordinates

\texttt{camLocalMin.Zc}

Z-coordinate of camera perspective center, always same units as XYZ object coordinates

\textbf{Reference}

Photogrammetry Toolbox Reference manual

\textbf{Remarks}

All angles must be in degrees. The input angles are redefined within the function to be \( \pm 180^\circ \). The input angles (possibly redefined within \( \pm 180^\circ \)) are echoed in the first elements of the fields of output structure \texttt{camLocalMin}. Elements 2 through 4 are the possible local minima that can occur for resection on nearly planar calibration target fields. The location of local and global minima in the nonlinear least squares solution for resection and the location of alternate solutions for nearly planar target fields is especially relevant to wind tunnel and solar sail applications since quite often targets on the object of interest are found to lie almost in a plane. One of the concerns of nonlinear least squares solutions such as used in space resection is that a local rather than a global minimum may have been reached. Whether or not a local minimum rather than the global minimum is reached is heavily dependent on the initial estimates of the coefficients. For cases where we have very good initial estimates of the exterior orientation of a camera, we arrive at the global minimum quite readily. However, for cases where it may be necessary to set all the initial estimates to zero (except possibly \( Z \)) it is then found that sometimes the solution converges to a local minimum for which the residuals are quite a bit larger than the global minimum. In other cases, especially for planar objects, the local minimum may have residuals that are within the range of the global minimum. For these local minimum the exterior orientation of the camera is incorrect. The function \texttt{resectionLocalMin} determines estimates of 3 such local minima so that one can then transform the possibly incorrect exterior orientation to improve the start values for a rerun the resection function.

\textbf{Example script}

\texttt{resectionLocalMinExample.m}

\textbf{Equations}

\[
\omega = \begin{bmatrix} -\omega & -\omega & \omega \end{bmatrix} \\
\phi = \begin{bmatrix} -\phi & -\phi & \phi \end{bmatrix} \\
\kappa = \begin{bmatrix} \kappa \pm 180 \end{bmatrix} \\
X_c = \begin{bmatrix} X_c & -X_c & -X_c \end{bmatrix} \\
Y_c = \begin{bmatrix} Y_c & -Y_c & -Y_c \end{bmatrix} \\
Z_c = \begin{bmatrix} (2Z - Z_c) & Z_c & (2Z - Z_c) \end{bmatrix}
\]

the smallest absolute value is taken for the \( \pm \) choice in the expressions (\( \kappa \pm 180 \)), which restricts the output value of \( \kappa \) to \( \pm 180^\circ \).
residual_exterior

Purpose
Estimation of residual of calculated image coordinates from measured ones for optimization of exterior orientation parameters

Syntax
[dd] = residual_exterior(ex_orien,in_orien1,in_orien2,camformat,xyimag,xyzobj)

Arguments
camformat
1-column array containing the following camera format data:
Number of horizontal pixels
Number of vertical pixels
Horizontal pixel spacing (mm/pixel)
Vertical pixel spacing (mm/pixel)

ex_orien
1-column array of the approximate exterior orientation parameters,
(ω, φ, κ, Xc, Yc, Zc)

in_orien1
1-column array of the first subset of the approximate interior orientation parameters,
(c, x_p, y_p, S_h / S_v, K_1)

in_orien2
1-column array of the second subset of the approximate interior orientation parameters,
(K_2, P_1, P_2)

xyimag
2-column array of the image coordinates (x, y) of a set of targets in pixels

xyzobj
3-column array of the object space coordinates (X, Y, Z) of a set of targets, and the units are consistent with (X_c, Y_c, Z_c) (typically in inches)

Output
dd
residual of the calculated image coordinates from the measured image coordinates of targets

Remarks
This function provides an objective function ‘dd’ for optimization for the exterior orientation parameters.

Called by
camcal_fun_1.m
Purpose
Estimation of residual of calculated image coordinates from measured ones for optimization of the first subset of interior orientation parameters

Syntax
\[ [dd] = \text{residual\_interior1}(\text{in\_orien1}, \text{ex\_orien}, \text{in\_orien2}, \text{camformat}, \text{xyimag}, \text{xyzobj}) \]

Arguments
- \text{camformat}
  1-column array containing the following camera format data:
  - Number of horizontal pixels
  - Number of vertical pixels
  - Horizontal pixel spacing (mm/pixel)
  - Vertical pixel spacing (mm/pixel)
- \text{ex\_orien}
  1-column array of the approximate exterior orientation parameters, 
  \((\kappa, \varphi, \psi, X_\xi, Y_\xi, Z_\xi)\)
- \text{in\_orien1}
  1-column array of the first subset of the approximate interior orientation parameters,
  \((c, x_p, y_p, S_h / S_v, K_1)\)
- \text{in\_orien2}
  1-column array of the second subset of the approximate interior orientation parameters,
  \((K_2, P_1, P_2)\)
- \text{xyimag}
  2-column array of the image coordinates \((x, y)\) of a set of targets in pixels
- \text{xyzobj}
  3-column array of the object space coordinates \((X, Y, Z)\) of a set of targets, and the units are consistent with \((X_\xi, Y_\xi, Z_\xi)\) (typically in inches)

Output
\(dd\)
residual of the calculated image coordinates from the measured image coordinates of targets

Remarks
This function provides an objective function ‘dd’ for optimization for the first subset of the interior orientation parameters.

Called by
\text{camcal\_fun\_1.m}
### residual_interior2

**Purpose**
Estimation of residual of calculated image coordinates from measured ones for optimization of the second subset of interior orientation parameters

**Syntax**
```
[dd] = residual_interior2(in_orien2,ex_orien,in_orien1,camformat,xyimag,xyzobj)
```

**Arguments**
- **camformat**
  1-column array containing the following camera format data:
  - Number of horizontal pixels
  - Number of vertical pixels
  - Horizontal pixel spacing (mm/pixel)
  - Vertical pixel spacing (mm/pixel)

- **ex_orien**
  1-column array of the approximate exterior orientation parameters,
  \((\alpha, \varphi, \kappa, X_e, Y_e, Z_e)\)

- **in_orien1**
  1-column array of the first subset of the approximate interior orientation parameters,
  \((c, x_p, y_p, S_h / S_v, K_v)\)

- **in_orien2**
  1-column array of the second subset of the approximate interior orientation parameters,
  \((K_v, P_1, P_2)\)

- **xyimag**
  2-column array of the image coordinates \((x, y)\) of a set of targets in pixels

- **xyzobj**
  3-column array of the object space coordinates \((X, Y, Z)\) of a set of targets, and the units are consistent with \((X_e, Y_e, Z_e)\) (typically in inches)

**Output**
- **dd**
  residual of the calculated image coordinates from the measured image coordinates of targets

**Remarks**
This function provides an objective function ‘dd’ for optimization for the second subset of the interior orientation parameters.

**Called by**
camcal_fun_1.m
**roiPolyselect**

**Purpose**
create an image that only contains the polygon region of interest (roi) selected or, optionally, has that regions removed (set to 0 grayscale)

**Syntax**
```matlab
imgOut = roiPolyselect(img, rejectFlag);
```

**Arguments**
- **img**
  image variable in the workspace or a valid image file name (either a character string or character variable). The 1st input argument `img` must be passed to the function in order to utilize the optional 2nd input argument `rejectFlag`.

- **rejectFlag**
  optional input argument entered as a character string (or character variable) set to `'reject'` to create an output image with the polygon roi set to 0 and the rest of the image to remain as is; any string other than `'reject'` will be ignored

**Output**
- **imgOut**
  image (same size and class as input image) with polygon roi data from `img` superimposed on a background of 0 (or if `rejectFlag` set to `'reject'` `imgOut` will be original image with polygon roi set to 0)

**Example script**
`roiPolyselectExample.m` with input files `Sample Files\image1.tif` and `Sample Files\image2.tif`

**Remarks**
The polygon roi is selected by positioning the cursor and clicking the left mouse button at each vertex of the polygon. Press ‘Enter’ to exit the function. The polygon is automatically closed to the 1st point selected. Note that every time the function `roiPolyselect` is invoked a new figure window is created. To remove the currently selected figure window, enter ‘`close`’ at the command line, or ‘`close all`’ to close all MATLAB figures. The function should always be followed by a semicolon ‘;’ to suppress printout of the output image `imgOut` to the Command Window. The 1st input argument must be passed to the function in order to utilize the optional 2nd input argument `rejectFlag`. 
roiSelect

Purpose
create an image that only contains the regions of interest (roi) selected or, optionally, has
those regions removed (set to 0 grayscale)

Syntax
[imgOut roi] = roiSelect(img, rejectFlag);

Arguments
img
image variable in the workspace or a valid image file name (either a character string or
character variable); Note that if the function is called without input arguments, an image file
dialog box opens from which the user can select the proper image file. The 1st input argument
img must be passed to the function in order to utilize the optional 2nd input argument
rejectFlag.

rejectFlag
optional input argument entered as a character string (or character variable) set to 'reject' to
create an output image with the roi's set to 0 and the rest of the image to remain as is; any
string other than 'reject' will be ignored

Output
imgOut
image (same size and class as input image) with roi data from img superimposed on a
background of 0 (or if rejectFlag set to 'reject' img Out will be original image with roi's set to
0)

roi
numeric N x 4 array containing [xmin ymin width height] for N roi's, one roi per row; the
corners of the roi are given by (xmin, ymin) and (xmin+width, ymin+height)

Example script
roiSelectExample.m with input files 'Sample Files\image1.tif' and 'Sample
Files\image2.tif'

Remarks
The rectangular roi is selected by positioning the cursor to one corner of the desired
rectangular area and then pressing the left mouse button and dragging to the other corner of
the rectangle. A single roi or many roi's can be selected. Press the left mouse button outside
the image to exit the function. Note that every time the function roiSelect is invoked a new
figure window is created. To remove the currently selected figure window, enter 'close'
at the command line, or 'close all' to close all MATLAB figures. The function should always
be followed by a semicolon ';' to suppress printout of the output image imgOut to the
Command Window. The 1st input argument must be passed to the function in order to utilize
the optional 2nd input argument rejectFlag.


Purpose
compute common $\omega, \phi, \kappa$ rotation matrix

Syntax
$m = \text{rotationMatrix}(\omega, \phi, \kappa, \text{AngleUnits})$

Arguments
omega
angle about $X$-axis, taken as $+$ for CCW rotation when viewing down the axis toward the origin; in degrees unless $\text{AngleUnits} = \text{‘radians’}$

phi
angle about $Y$-axis, taken as $+$ for CCW rotation when viewing down the axis toward the origin; in degrees unless $\text{AngleUnits} = \text{‘radians’}$

kappa
angle about $Z$-axis, taken as $+$ for CCW rotation when viewing down the axis toward the origin; in degrees unless $\text{AngleUnits} = \text{‘radians’}$

AngleUnits
optional argument to force units to radians with $\text{AngleUnits} = \text{‘radians’}$; if left off (using only 3 arguments) or set to anything other than ‘radians’, units of degrees will be assumed; for example if $\text{AngleUnits} = \text{‘radian’}$ then the exact match is not met and the units of degrees will be assumed

Output
$m$
output is a $3 \times 3$ array of the $\omega, \phi, \kappa$ rotation matrix

output for $m = \text{rotationMatrix}(0, 0, 0)$
$m =$
\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

output for $m = \text{rotationMatrix}(90, 90, 90)$
$m =$
\[
\begin{bmatrix}
0.0000 & 0.0000 & 1.0000 \\
-0.0000 & -1.0000 & 0.0000 \\
1.0000 & -0.0000 & 0.0000
\end{bmatrix}
\]

output for $m = \text{rotationMatrix}(\pi/2, \pi/2, \pi/2, \text{‘radians’})$
$m =$
\[
\begin{bmatrix}
0.0000 & 0.0000 & 1.0000 \\
-0.0000 & -1.0000 & 0.0000 \\
1.0000 & -0.0000 & 0.0000
\end{bmatrix}
\]

Reference
Remarks
order of application of angles is omega, phi, and then kappa

Example script
rotationMatrixExample.m

Equations
\[
\begin{align*}
    m_{11} &= \cos \phi \cos \kappa \\
    m_{12} &= \sin \omega \sin \phi \cos \kappa + \cos \omega \sin \kappa \\
    m_{13} &= -\cos \omega \sin \phi \cos \kappa + \sin \omega \sin \kappa \\
    m_{21} &= -\cos \phi \sin \kappa \\
    m_{22} &= -\sin \omega \sin \phi \sin \kappa + \cos \omega \cos \kappa \\
    m_{23} &= \cos \omega \sin \phi \sin \kappa + \sin \omega \cos \kappa \\
    m_{31} &= \sin \phi \\
    m_{32} &= -\sin \omega \cos \phi \\
    m_{33} &= \cos \omega \cos \phi
\end{align*}
\]
Compute rotation matrix in terms of azimuth, elevation, roll

**Syntax**

\[ m = \text{rotationMatrixAzElevationRoll}(\text{Azimuth}, \text{Elevation}, \text{Roll}, \text{AngleUnits}) \]

**Arguments**

- **Azimuth**
  angle about \( Z \)-axis, taken as + for CW rotation; in degrees unless \( \text{AngleUnits} = \text{`radians`} \)

- **Elevation**
  angle about new \( Y \)-axis formed after the azimuth rotation, taken as + for CCW; in degrees unless \( \text{AngleUnits} = \text{`radians`} \)

- **Roll**
  angle about the new \( X \)-axis formed after the azimuth and Elevation rotations, taken as + for CCW rotation; in degrees unless \( \text{AngleUnits} = \text{`radians`} \)

- **AngleUnits**
  optional argument to force units to radians with \( \text{AngleUnits} = \text{`radians`} \); if left off (using only 3 arguments) or set to anything other than ‘radians’, units of degrees will be assumed; for example if \( \text{AngleUnits} = \text{`radian`} \) then the exact match is not met and the units of degrees will be assumed

**Output**

\( m \)

output is a \( 3 \times 3 \) array of the rotation matrix using Azimuth, Elevation, and Roll

- output for \( m = \text{rotationMatrixAzElevationRoll}(0, 0, 0) \)
  \[
  m = \\
  \begin{bmatrix}
  1 & 0 & 0 \\
  0 & 0 & 1 \\
  0 & -1 & 0 \\
  \end{bmatrix}
  \]

- output for \( m = \text{rotationMatrixAzElevationRoll}(90, 90, 90) \)
  \[
  m = \\
  \begin{bmatrix}
  1.0000 & 0 & 0.0000 \\
  0 & -1.0000 & 0.0000 \\
  0.0000 & -0.0000 & -1.0000 \\
  \end{bmatrix}
  \]

- output for \( m = \text{rotationMatrixAzElevationRoll}(\pi/2, \pi/2, \pi/2, \text{`radians'}) \)
  \[
  m = \\
  \begin{bmatrix}
  1.0000 & 0 & 0.0000 \\
  0 & -1.0000 & 0.0000 \\
  0.0000 & -0.0000 & -1.0000 \\
  \end{bmatrix}
  \]

**Reference**

order of application of angles is azimuth, Elevation, and then Roll
Example script

rotationMatrixAzElevationRollExample.m

Equations

\[
\begin{align*}
    m_{11} &= \sin \alpha \sin e \sin r + \cos \alpha \cos r \\
    m_{12} &= -\cos \alpha \sin e \sin r + \sin \alpha \cos r \\
    m_{13} &= \cos e \sin r \\
    m_{21} &= \sin \alpha \sin e \cos r - \cos \alpha \sin r \\
    m_{22} &= -\cos \alpha \sin e \cos r - \sin \alpha \sin r \\
    m_{23} &= \cos e \cos r \\
    m_{31} &= \sin \alpha \cos e \\
    m_{32} &= -\cos \alpha \cos e \\
    m_{33} &= -\sin e
\end{align*}
\]

where $\alpha =$ azimuth, $e =$ elevation, $r =$ roll
**rotationMatrixAzTiltSwing**

**Purpose**
compute rotation matrix in terms of azimuth, tilt, swing

**Syntax**
m = rotationMatrixAzTiltSwing (Azimuth, Tilt, Swing, AngleUnits)

**Arguments**

Azimuth
angle about Z-axis, taken as + for CW rotation; in degrees unless AngleUnits = 'radians'

Tilt
angle about new X-axis formed after the azimuth rotation, taken as + for CCW; in degrees unless AngleUnits = 'radians'

Swing
angle about the new Z-axis formed after the azimuth and tilt rotations, taken as + for CCW rotation; in degrees unless AngleUnits = 'radians'

AngleUnits
optional argument to force units to radians with AngleUnits = 'radians'; if left off (using only 3 arguments) or set to anything other than 'radians', units of degrees will be assumed; for example if AngleUnits = 'radian' then the exact match is not met and the units of degrees will be assumed

**Output**
m
output is a 3 × 3 array of the α, φ, κ rotation matrix

output for m = rotationMatrixAzTiltSwing (0, 0, 0)
m =

\[
\begin{pmatrix}
-1 & 0 & 0 \\
0 & -1 & 0 \\
0 & 0 & 1 \\
\end{pmatrix}
\]

output for m = rotationMatrixAzTiltSwing (90, 90, 90)
m =

\[
\begin{pmatrix}
0.0000 & 0.0000 & -1.0000 \\
-1.0000 & -0.0000 & -0.0000 \\
-1.0000 & 0.0000 & 0.0000 \\
\end{pmatrix}
\]

output for m = rotationMatrixAzTiltSwing (\pi/2, \pi/2, \pi/2, 'radians')
m =

\[
\begin{pmatrix}
0.0000 & 0.0000 & -1.0000 \\
-1.0000 & -0.0000 & -0.0000 \\
-1.0000 & -0.0000 & 0.0000 \\
\end{pmatrix}
\]

**Reference**

**Remarks**
order of application of angles is azimuth, tilt, and then swing

**Example script**
rotationMatrixAzTiltSwingExample.m

122
Equations

\[ m_{11} = -\cos \alpha \cos s - \sin \alpha \cos t \sin s \]
\[ m_{12} = \sin \alpha \cos s - \cos \alpha \cos t \sin s \]
\[ m_{13} = -\sin t \sin s \]
\[ m_{21} = \cos \alpha \sin s - \sin \alpha \cos t \cos s \]
\[ m_{22} = -\sin \alpha \sin s - \cos \alpha \cos t \cos s \]
\[ m_{23} = -\sin t \cos s \]
\[ m_{31} = -\sin \alpha \sin t \]
\[ m_{32} = -\cos \alpha \sin t \]
\[ m_{33} = \cos t \]

where \( \alpha = \) azimuth, \( t = \) tilt, \( s = \) swing
rotationMatrixDuality

Purpose
outputs alternate set (duality) of $\omega, \phi, \kappa$ that has identical rotation matrix as computed with input $\omega, \phi, \kappa$

Syntax
$[\omega_{\text{Dual}}, \phi_{\text{Dual}}, \kappa_{\text{Dual}}] = \text{rotationMatrixDuality}(\omega, \phi, \kappa, \text{AngleUnits})$

Arguments
omega
angle about X-axis, taken as + for CCW rotation when viewing down the axis toward the origin; in degrees unless \text{AngleUnits} = ‘radians’

phi
angle about Y-axis, taken as + for CCW rotation when viewing down the axis toward the origin; in degrees unless \text{AngleUnits} = ‘radians’

kappa
angle about Z-axis, taken as + for CCW rotation when viewing down the axis toward the origin; in degrees unless \text{AngleUnits} = ‘radians’

AngleUnits
optional argument to force units to radians with \text{AngleUnits} = ‘radians’; if left off (using only 3 arguments) or set to anything other than ‘radians’ (exactly), units of degrees will be assumed; for example if \text{AngleUnits} = ‘radian’ then the exact match is not met and the units of degrees will be assumed

Output
omegaDual
angle about X-axis, taken as + for CCW rotation when viewing down the axis toward the origin; in degrees unless \text{AngleUnits} = ‘radians’

phiDual
angle about Y-axis, taken as + for CCW rotation when viewing down the axis toward the origin; in degrees unless \text{AngleUnits} = ‘radians’

kappaDual
angle about Z-axis, taken as + for CCW rotation when viewing down the axis toward the origin; in degrees unless \text{AngleUnits} = ‘radians’

output for $[\omega_{\text{Dual}}, \phi_{\text{Dual}}, \kappa_{\text{Dual}}] = \text{rotationMatrixDuality}(0, 0, 0)$
$[180, 180, 180]$

output for $[\omega_{\text{Dual}}, \phi_{\text{Dual}}, \kappa_{\text{Dual}}] = \text{rotationMatrixDuality}(0, 0, 0, \text{‘radians’})$
$[3.1416, 3.1416, 3.1416]$

output for $[\omega_{\text{Dual}}, \phi_{\text{Dual}}, \kappa_{\text{Dual}}] = \text{rotationMatrixDuality}(10, -20, 30)$
$[-170, -160, -150]$

Reference
PE&RS vol. 56 No.9, Sept. 1990, pp. 1281-1283

Remarks
order of application of Euler angles is omega, phi, and then kappa. Note that the duality of the rotation matrix is not simply due to the cyclical nature of the trigonometric functions with
additions of ± 2π. Additions of ± 2π actually produce the same angle, unlike the duality angles which differ by ± π. The set of duality angles is an alternate way to angularly position a camera to the same final angular orientation as the matching set of angles. This function should help in interpretation of space resection results in which the computed angles appear quite different from expected (or from other solutions), but actually produce the exact same rotation matrix (final position) and are thus fully equivalent.

Example script
rotationMatrixDualityExample.m

Equations

\[
\omega_{\text{Dual}} = \omega + \pi \text{ or } \omega - \pi \\
\phi_{\text{Dual}} = \pi - \phi \text{ or } -\phi - \pi \\
\kappa_{\text{Dual}} = \kappa + \pi \text{ or } \kappa - \pi
\]

The minimum of the absolute values of either of the 2 choices for each of \(\omega_{\text{Dual}}\), \(\phi_{\text{Dual}}\), or \(\kappa_{\text{Dual}}\) is selected as the output value for each angle.
saveCamStruct

Purpose
Saves camera parameter structure in a text file for later loading into a script or function with the matching function loadCamStruct.

Syntax
saveCamStruct(fileName, camStructure)

Arguments

fileName
fileName of file to save camera parameter structure (such as the string 'fileName' or the string variable filename)

camStructure
camera parameter structure with fields as follows:

camStructure.c
principal distance $c$ (or camera constant), usually mm

camStructure.xp
$x$-value of the photogrammetric principal point, usually mm, but always same units as $c$.

camStructure.yp
$y$-value of the photogrammetric principal point, usually mm, but always same units as $c$.

camStructure.m
$3 \times 3$ rotation matrix, usually from function rotationMatrix

camStructure.Xc
$X$-coordinate of camera perspective center, always same units as XYZ object coordinates

camStructure.Yc
$Y$-coordinate of camera perspective center, always same units as XYZ object coordinates

camStructure.Zc
$Z$-coordinate of camera perspective center, always same units as XYZ object coordinates

Output
text file with name filename (which may contain the path) like:
c = 25.0000
xp =  0.50000
yp = -0.50000
m =  0.4924038765061041
m = -0.5868240888334652
m =  0.6427876096865393
m =  0.8700019037522058
m =  0.3104684609733676
m = -0.383022215594890
m =  0.0252013862574872
m =  0.7478280708194912
m =  0.6634139481689384
Xc =  10.00000
Yc =  20.00000

126
**Remarks**

`saveCamStruct` is a simple function to save the basic camera parameter structure in a human readable text file. When saved in this format the matching function `loadCamStruct` can be used to load the camera parameter structure into a structure variable within a script or function for further application. The rotation matrix $m$ is saved in row order (default for MATLAB) in the order $m_{11}$, $m_{21}$, $m_{31}$, $m_{22}$, $m_{23}$, $m_{32}$, $m_{33}$. Note that this simple function ignores other fields of the camera parameter structure other than those identified above. The current version of this simple function has minimal error handling.

**Example script**

`saveCamStructExample.m` with output to 'Sample Files\camStruct.txt'
**Purpose**
Single view photogrammetry determinations of 1 or 2 coordinates with 2 or 1 of the other coordinates known respectively. Can solve singly for coordinates \( X, Y, \) or \( Z \) if the other 2 coordinates are known. Also can solve for coordinate pairs \( X & Y, X & Z, \) or \( Y & Z \) if the other coordinate of the triplet is known.

**Syntax**

\[
XYZsv = \text{singleView}(\text{cam}, \text{ xmm}, \text{XYZ})
\]

**Arguments**

- `cam`
  structure with fields as follows:
  - `cam.c`
    principal distance \( c \) (or camera constant), usually \( mm \)
  - `cam.xp`
    \( x \)-value of the photogrammetric principal point, usually \( mm \), but always same units as \( c \).
  - `cam.yp`
    \( y \)-value of the photogrammetric principal point, usually \( mm \), but always same units as \( c \).
  - `cam.m`
    \( 3 \times 3 \) rotation matrix, usually from function `rotationMatrix`
  - `cam.Xc`
    \( X \)-coordinate of camera perspective center, always same units as \( XYZ \) object coordinates
  - `cam.Yc`
    \( Y \)-coordinate of camera perspective center, always same units as \( XYZ \) object coordinates
  - `cam.Zc`
    \( Z \)-coordinate of camera perspective center, always same units as \( XYZ \) object coordinates

- `xmm`
  \( N \times 3 \) array with point numbers in 1\(^{st} \) column. The array `xmm` is of the form:

\[
pt_1 \ x_1 \ y_2
pt_2 \ x_2 \ y_2
\ldots
\ldots
pt_N \ x_N \ y_N
\]

- `XYZ`
  structure with the following 4 fields:
  - `XYZ.pnt` (target number)
  - `XYZ.X` (\( X \)-value in object space)
  - `XYZ.Y` (\( Y \)-value in object space)
  - `XYZ.Z` (\( Z \)-value in object space)
the coordinate(s) to be solved for should be set to [] in XYZ structure as, for instance, $X_{Y} = [;$. The other coordinates not solved for are echoed in the output array $XYZsv$ along with the coordinates solved for. Units of output array $XYZsv$ same as units of $XYZ$ (and $Xc, Yc, Zc$)

**Output**

$XYZsv$

output is an $N \times 4$ or $N \times 5$ array with point numbers taken from target numbers which are common to both $xymm$ and $XYZ$ structure. There can be missing target numbers in either the image or object coordinate input arguments. Only data from targets common to both are outputted to $XYZsv$. The output array $XYZsv$ is of the form below (except for single coordinate solutions of $X, Y$, or $Z$ only where a 5th column is also outputted containing the standard deviation as determined by least squares of the single coordinate solved for):

$$
\begin{array}{cccc}
pt_1 & x_1 & y_1 & z_1 & (\sigma_1) \\
pt_2 & x_2 & y_2 & z_2 & (\sigma_2) \\
.  \\
.  \\
pt_N & x_N & y_N & z_N & (\sigma_N)
\end{array}
$$

**Remarks**

This function can solve for single coordinates if the other 2 coordinates are known, or can solve for 2 coordinates if only 1 other coordinate is known from a single view. The camera parameters listed above for the structure $cam$ most all be known along with image coordinates corresponding to the object coordinates. Cases where 2 coordinates are known and one is solve for result in 2 equations (collinearity equations, 1 for x-image, 1 for y-image) in 1 unknown. Thus least squares can be used to determine the single coordinate while also computing as estimate of the standard deviation of the coordinate (but with only 1 degree of freedom). For those cases the 5th column of the output array $XYZsv$ contains the estimated standard deviation from the least squares computation.

**Example script**

singleViewExample.m

**Equations**

$$
x = x_p - c \left[ \frac{m_{11}(X-X_c) + m_{12}(Y-Y_c) + m_{13}(Z-Z_c)}{m_{31}(X-X_c) + m_{32}(Y-Y_c) + m_{33}(Z-Z_c)} \right]
$$

$$
y = y_p - c \left[ \frac{m_{21}(X-X_c) + m_{22}(Y-Y_c) + m_{23}(Z-Z_c)}{m_{31}(X-X_c) + m_{32}(Y-Y_c) + m_{33}(Z-Z_c)} \right]
$$

the collinearity equations above can be recast in the following form

$$
a_1 X + a_2 Y + a_3 Z = a_4 X_c + a_5 Y_c + a_6 Z_c
$$

$$
a_4 X + a_5 Y + a_6 Z = a_1 X_c + a_2 Y_c + a_3 Z_c
$$

where
\[ a_1 = (x - x_p) m_{31} + c m_{11} \]
\[ a_2 = (x - x_p) m_{32} + c m_{12} \]
\[ a_3 = (x - x_p) m_{33} + c m_{13} \]
\[ a_4 = (y - y_p) m_{31} + c m_{21} \]
\[ a_5 = (y - y_p) m_{32} + c m_{22} \]
\[ a_6 = (y - y_p) m_{33} + c m_{23} \]

**X, Y solution:**

\[
A = \begin{bmatrix} a_1 & a_2 \\ a_4 & a_5 \end{bmatrix}
\]

\[
B = \begin{bmatrix} a_1 X_c + a_2 Y_c + a_4 Z_c - a_5 Z_a \\ a_4 X_c + a_5 Y_c + a_6 Z_c - a_2 Z_a \end{bmatrix}
\]

\[
\begin{bmatrix} X \\ Y \end{bmatrix} = A \backslash B
\]

where \( A \backslash B \) is the MATLAB operator for Gaussian elimination, or if over-determined, for linear least squares

**X, Z solution:**

\[
A = \begin{bmatrix} a_1 & a_3 \\ a_4 & a_6 \end{bmatrix}
\]

\[
B = \begin{bmatrix} a_1 X_c + a_3 Y_c + a_4 Z_c - a_6 Y_c \\ a_4 X_c + a_6 Y_c + a_5 Z_c - a_3 Y_c \end{bmatrix}
\]

\[
\begin{bmatrix} X \\ Y \end{bmatrix} = A \backslash B
\]

**Y, Z solution:**

\[
A = \begin{bmatrix} a_2 & a_3 \\ a_5 & a_6 \end{bmatrix}
\]

\[
B = \begin{bmatrix} a_2 X_c + a_3 Y_c + a_5 Z_c - a_6 X_c \\ a_5 X_c + a_6 Y_c + a_4 Z_c - a_2 X_c \end{bmatrix}
\]

\[
\begin{bmatrix} X \\ Y \end{bmatrix} = A \backslash B
\]
\( X \) solution:

\[
A = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}
\]

\[
B = \begin{bmatrix} a_1 x_c + a_2 y_c + a_3 z_c - a_2 y - a_3 z \\ a_4 x_c + a_5 y_c + a_6 z_c - a_4 y - a_6 z \end{bmatrix}
\]

\[
[X] = A \backslash B
\]

\( Y \) solution:

\[
A = \begin{bmatrix} a_2 \\ a_3 \end{bmatrix}
\]

\[
B = \begin{bmatrix} a_1 x_c + a_2 y_c + a_3 z_c - a_1 x - a_3 y \\ a_4 x_c + a_5 y_c + a_6 z_c - a_4 x - a_6 y \end{bmatrix}
\]

\[
[Y] = A \backslash B
\]

\( Z \) solution:

\[
A = \begin{bmatrix} a_3 \\ a_6 \end{bmatrix}
\]

\[
B = \begin{bmatrix} a_1 x_c + a_2 y_c + a_3 z_c - a_2 x - a_3 y \\ a_4 x_c + a_5 y_c + a_6 z_c - a_4 y - a_5 z \end{bmatrix}
\]

\[
[Z] = A \backslash B
\]

Computation of standard deviation for \( X, Y, \) or \( Z \) single coordinate least squares solution (with \( X \) replaced by \( Y \) or \( Z \) as necessary):

\[
V = [A][X] - [B]
\]

\[
S_0 = \sqrt{V^T V}
\]

\[
cov = \left( [A^T][A] \right)^t
\]

\[
\sigma = S_0 \sqrt{cov}
\]

where \( V \) is a column vector of residuals, \( S_0 \) is the standard deviation of unit weight, \( cov \) is the covariance matrix and \( \sigma \) is the estimate of the standard deviation of either \( X, Y, \) or \( Z \) from least squares estimation.
TransposeAngles

Purpose returns $\omega_T, \phi_T, \kappa_T$ for a rotation matrix that is the transpose of the rotation matrix formed by the input arguments $\omega, \phi, \kappa$.

Syntax

$$\text{Angle} = \text{TransposeAngles}(\text{Parameter})$$

Arguments

Parameter structure with the following fields:

Parameter.omega
angle in degrees about $X$-axis, $\omega$ taken as + for CCW rotation when viewing down the axis toward the origin

Parameter.phi
angle in degrees about $Y$-axis, $\phi$ taken as + for CCW rotation when viewing down the axis toward the origin

Parameter.kappa
angle in degrees about $Z$-axis, $\kappa$ taken as + for CCW rotation when viewing down the axis toward the origin

Output

Angle structure with the following fields:

Angle.omega
angle in degrees about $X$-axis, $\omega_T$ taken as + for CCW rotation when viewing down the axis toward the origin

Angle.phi
angle in degrees about $Y$-axis, $\phi_T$ taken as + for CCW rotation when viewing down the axis toward the origin

Angle.kappa
angle in degrees about $Z$-axis, $\kappa_T$ taken as + for CCW rotation when viewing down the axis toward the origin

Reference


Remarks

order of application of angles is omega, phi, and then kappa. This function can be useful for cases where the solution is desired in terms of the transpose of the rotation matrix, but the solution in hand is in terms of the rotation matrix without transpose (for example when using the function conformal3DNLLS). Note that the angles $\omega_T$ and $\kappa_T$ should not be found from the diagonal elements of the rotation matrix $m_{33}$ and $m_{11}$ since the cosine function returns the same value for ± angles, thus the signs of $\omega$ and $\kappa$ may not be correctly determined using those elements. Also note that the 4-quadrant inverse tangent $\text{atan2}$ is used in the function to determine $\omega_T$ and $\kappa_T$ and that the output angle $\phi_T$ is limited by the $\text{asind}$ function to ± 90°. A warning error message test (for maximum absolute error > $10^{-12}$) is built into the function.
which compares the rotation matrix generated from the output angles to the transpose of the rotation matrix generated from the input angles.

**Example script**  
TransposeAnglesExample.m

**Equations**

the rotation matrix for the input angles $\omega, \phi, \kappa$ is given by

\[
\begin{align*}
    m_{11} &= \cos \phi \cos \kappa \\
    m_{12} &= \sin \omega \sin \phi \cos \kappa + \cos \omega \sin \kappa \\
    m_{13} &= -\cos \omega \sin \phi \cos \kappa + \sin \omega \sin \kappa \\
    m_{21} &= -\cos \phi \sin \kappa \\
    m_{22} &= -\sin \omega \sin \phi \sin \kappa + \cos \omega \cos \kappa \\
    m_{23} &= \cos \omega \sin \phi \sin \kappa + \sin \omega \cos \kappa \\
    m_{31} &= \sin \phi \\
    m_{32} &= -\sin \omega \cos \phi \\
    m_{33} &= \cos \omega \cos \phi
\end{align*}
\]

the output angles $\omega_t, \phi_t, \kappa_t$ are found from the rotation matrix formed from the input angles $\omega, \phi, \kappa$ using the following equations

\[
\begin{align*}
    \phi_t &= \sin^{-1} m_{33} \\
    \omega_t &= -\tan^{-1} \left( \frac{m_{23}}{m_{33}} \right) \\
    \kappa_t &= -\tan^{-1} \left( \frac{m_{22}}{m_{33}} \right)
\end{align*}
\]

the rotation matrix formed from the function output angles $\omega_t, \phi_t, \kappa_t$ equals the transpose of the rotation matrix formed from the input angles $\omega, \phi, \kappa$.
xy2XYZ

**Purpose**
Determination of object-space coordinates (X, Y, Z) of a target from the corresponding image coordinates in two images (A and B) by photogrammetric intersection

**Syntax**
\[
\begin{align*}
\text{[Xtarg, Ytarg, Ztarg]} = \\
\text{xy2XYZ(xPixA, yPixA, xPixB, yPixB, oriA, oriB, camformatA, camformatB)}
\end{align*}
\]

**Arguments**
- \((xPixA, yPixA)\)
  image coordinates in image A in pixels
- \((xPixB, yPixB)\)
  image coordinates in image B in pixels
- \(\text{oriA}\)
  1-column array of the orientation parameters for Camera A \((\omega, \phi, \kappa, X_c, Y_c, Z_c)\) and \((c, x_p, y_p, S_h / S_v, K_1, K_2, P_1, P_2)\)
- \(\text{oriB}\)
  1-column array of the orientation parameters for Camera A \((\omega, \phi, \kappa, X_c, Y_c, Z_c)\) and \((c, x_p, y_p, S_h / S_v, K_1, K_2, P_1, P_2)\)
- \(\text{camformatA}\)
  1-column array containing the following camera format data for Camera A:
  - Number of horizontal pixels
  - Number of vertical pixels
  - Horizontal pixel spacing (mm/pixel)
  - Vertical pixel spacing (mm/pixel)
- \(\text{camformatB}\)
  1-column array containing the following camera format data for Camera B:
  - Number of horizontal pixels
  - Number of vertical pixels
  - Horizontal pixel spacing (mm/pixel)
  - Vertical pixel spacing (mm/pixel)

**Output**
\[
\text{[Xtarg, Ytarg, Ztarg]} \text{ object-space coordinates of a target}
\]

**Remarks**
This function is used for stereo photogrammetric measurements to determine the 3D object-space coordinates from two images.

**Example script**
xy2XYZExample.m

**Equations**
The detailed description of intersection is given in the following reference.
**xy2XZ**

| **Purpose** | Determination of object-space coordinates (X, Z) of a target from the corresponding image coordinates in one image for a given Y-coordinate |
| **Syntax** | \([X_{\text{targ}}, Z_{\text{targ}}] = \text{xy2XZ}(x_{\text{Pix}}, y_{\text{Pix}}, Y_{\text{targ}}, \text{ori}, \text{camformat})\) |
| **Arguments** | \((x_{\text{Pix}}, y_{\text{Pix}})\) image coordinates of a target in image in pixels  
\(Y_{\text{targ}}\) Y-coordinates in object space, the unit is consistent with \((X_c, Y_c, Z_c)\)  
\(\text{ori}\) 1-column array of the orientation parameters for camera \((\alpha, \varphi, \kappa, X_c, Y_c, Z_c)\) and \((c, x_p, y_p, S_h / S_v, K_1, K_2, P_1, P_2)\)  
\(\text{camformat}\) 1-column array containing the following camera format data for: \(\text{Number of horizontal pixels, Number of vertical pixels, Horizontal pixel spacing (mm/pixel), Vertical pixel spacing (mm/pixel)}\) |
| **Output** | \([X_{\text{targ}}, Z_{\text{targ}}]\) object-space coordinates (X, Z) of a target |
| **Remarks** | This single-camera method is a constrained intersection, which is particularly useful in wing deformation measurements. |
| **Example script** | xy2XZExample.m |
| **Equations** | The detailed description of this single-camera method is given in the following reference.  
xyplot

**Purpose**
Graphical comparison of measured image coordinates with calculated image coordinates from object-space coordinates of targets through projection (collinearity equations)

**Syntax**
xyplot(camformat, orien, xyimag, xyzobj, plot_No)

**Arguments**

- **orien**
  1-column array of the orientation parameters for camera \((\omega, \varphi, \kappa, X_c, Y_c, Z_c)\) and \((c, x_p, y_p, S_x, S_y, K_x, K_y, P_x, P_y)\)

- **xyzobj**
  object space coordinates \((X, Y, Z)\) of targets, and the units are consistent with \((X_c, Y_c, Z_c)\)
  (typically in inches)

- **camformat**
  1-column array containing the following camera format data for:
  - Number of horizontal pixels
  - Number of vertical pixels
  - Horizontal pixel spacing (mm/pixel)
  - Vertical pixel spacing (mm/pixel)

- **xyimag**
  measured image coordinates \((x, y)\) of targets in pixels

- **plot_No**
  plot number

**Output**
comparison plot of image coordinates (in mm) of targets

**Remarks**
This is a plotting function for comparison between measured and calculated images coordinates.

**Called by**
dlt0.m, dlt.m, camcal_fun.m

**Equations**
The detailed description of the collinearity equations is given in the following reference.

XYZ2xy

Purpose
Determination of image coordinates (x, y) from object-space coordinates (X, Y, Z) of a target through projection (collinearity equations)

Syntax
[xyimag]=XYZ2xy(ori,xyzobj,camformat)

Arguments
ori
1-column array of the orientation parameters for camera (ω, φ, κ, Xc, Yc, Zc) and (c, x0, y0, Sz / Sx, Ky, Kx, Px, Py, Pz).

xyzobj
object space coordinates (X, Y, Z) of a target, and the units are consistent with (Xc, Yc, Zc) (typically in inches)

camformat
1-column array containing the following camera format data for:

Number of horizontal pixels
Number of vertical pixels
Horizontal pixel spacing (mm/pixel)
Vertical pixel spacing (mm/pixel)

Output
[xyimag]
image coordinates (x, y) of a target in pixels

Remarks
This is a projection function for the given camera orientation parameters.

Example script
xy2XZExample.m

Equations
The detailed description of the collinearity equations is given in the following reference.

Specialized photogrammetric and image processing MATLAB functions useful for wind tunnel and other ground-based testing of aerospace structures are described. These functions include single view and multi-view photogrammetric solutions, basic image processing to determine image coordinates, 2D and 3D coordinate transformations and least squares solutions, spatial and radiometric camera calibration, epipolar relations, and various supporting utility functions.