PyTrx

Release 1.1

Penelope How, Nick Hulton, Lynne Buie

Oct 08, 2021

CONTENTS:

1 Quickstart **3**
- 1.1 PyTrx set-up . . . 3
- 1.2 Downloading PyTrx from GitHub . . . 3
- 1.3 Installing PyTrx through pip . . . 3
- 1.4 PyTrx Structure . . . 4

2 PyTrx Guide **7**
- 2.1 CamEnv . . . 7
- 2.2 DEM . . . 7
- 2.3 FileHandler . . . 7
- 2.4 Images . . . 8
- 2.5 Velocity . . . 8
- 2.6 Area . . . 8
- 2.7 Line . . . 8
- 2.8 Utilities . . . 8

3 PyTrx Modules **9**
- 3.1 Area module . . . 9
- 3.2 CamEnv module . . . 12
- 3.3 DEM module . . . 18
- 3.4 FileHandler module . . . 20
- 3.5 Images module . . . 25
- 3.6 Line module . . . 28
- 3.7 Utilities module . . . 29
- 3.8 Velocity module . . . 32

4 PyTrx Examples **41**
- 4.1 Automated area detection . . . 41
- 4.2 Manual area detection . . . 41
- 4.3 Manual line detection . . . 41
- 4.4 Point georectification . . . 42
- 4.5 Sparse feature-tracking . . . 42
- 4.6 Dense feature-tracking . . . 42
- 4.7 Sparse and dense feature-tracking . . . 42

5 Links and Acknowledgements **43**
- 5.1 PyTrx citations . . . 43
- 5.2 Permissions . . . 43
- 5.3 Acknowledgements . . . 44
- 5.4 Links . . . 44

6 Indices and tables 45

Python Module Index 47

Index 49

PyTrx (short for 'Python Tracking') is a Python object-oriented toolbox created for the purpose of calculating real-world measurements from oblique images and time-lapse image series. Its primary purpose is to obtain velocities, surface areas, and distances from imagery of glacial environments.

Authors: Penelope How (how@geus.dk), Nick Hulton, and Lynne Buie (née Addison)

CHAPTER

ONE

QUICKSTART

1.1 PyTrx set-up

PyTrx has been coded with Python 3 and has been tested on Linux and Windows operating systems (it should also work on Apple operating systems too, it just hasn't been tested). PyTrx was originally written using a Linux operating system, so the inputted file path structures given in the example scripts may differ between operating systems and it is therefore advised to check file path structures before running these.

PyTrx v1.1 can either be downloaded directly from the GitHub repository, or installed PyPI package manager (pip).

1.2 Downloading PyTrx from GitHub

PyTrx can be downloaded directly through the 'clone or download' icon on PyTrx's GitHub repository. To use PyTrx, you will need a working distribution of Python and the following key packages, which PyTrx strongly depends on:

- OpenCV (v3 and above): https://opencv.org
- GDAL (v2 and above): https://gisinternals.com
- Pillow (PIL) (v5 and above): https://pythonware.com

Be aware that these dependencies may not necessarily be installed with your distribution of Python (e.g. PythonXY, Anaconda), so you may have to install them separately. The .yml environment file provided in the GitHub repository can be used to set up an environment that holds all of the necessary Python packages to run PyTrx.

PyTrx has been tried and tested with the following dependency version configuration: *OpenCV=3.4.2*, *GDAL=2.3.2*, and *PIL=5.3*. PyTrx also needs other packages, which are commonly included with distributions of Python: *datetime*, *glob*, *imghdr*, *math*, *Matplotlib*, *NumPy*, *operator*, *os*, *pathlib*, *PyLab*, *SciPy*, *struct*, and *sys*. Compatibility with all newer versions of these packages are highly likely.

1.3 Installing PyTrx through pip

PyTrx is available through pip and can be installed with the following simple command:

```
pip install pytrx
```

WARNING There are difficulties with the GDAL package on pip, meaning that GDAL could not be declared explicitly as a PyTrx dependency. Please ensure that GDAL is installed separately if installing PyTrx through pip.

To check that PyTrx is working, open a Python console or IDE such as Spyder, type 'import PyTrx' and hit enter, followed by 'help(PyTrx)'. If PyTrx is working correctly, this should print PyTrx's metadata, including PyTrx's license,

a brief description of the toolset, and its structure. If this does not work and throws up an error, it is likely that the package dependencies are invalid so reconfigure them and then try again. Now you are all set up to use PyTrx.

1.4 PyTrx Structure

Detailed documentation is included in the scripts that make up PyTrx. Each script contains classes and functions for handling each aspect needed for photogrammetric processing.

For beginners in programming, it is advised to look at the example applications provided and adapt them accordingly for your own use. For experienced programmers, get stuck in. Feel free to contact us if you run into major problems or have constructive comments that will help us further PyTrx and its capabilities. We will not respond to minor troubleshooting or unconstructive comments.

1.4.1 CamEnv.py

Handles the associated data with the camera environment. The GCPs class handles the Ground Control Points (GCPs) and their correspondence to the associated DEM and CamImage object. The CamCalib class handles information concerning the camera calibration, i.e. the intrinsic camera matrix and lens distortion coefficients. This class contains functionality for reading in calibration files from .txt and .mat formats. The CamEnv compiles all the information about the camera environment from the GCPs and CamCalib classes, and also contains information about the camera object (pose and location). This is also where georectification functionality is held, with functions for projection and inverse projection. The class is initialised using a .txt file containing file path directories to all the associated data files.

1.4.2 DEM.py

Handles the DEM data. This currently supports .mat and .tif file types. The ExplicitRaster class represents a DEM as a numeric raster with explicit XY cell referencing in each grid cell. The class includes functions for densification, calculating viewsheds, and incorporates unbound functions that import a DEM file from .mat and .tif formats.

1.4.3 FileHandler.py

This module contains a set of functions for reading in data from files (such as image data and calibration information) and writing out data.

1.4.4 Images.py

Handles the image data, and the image sequence. The CamImage class holds information about a singular image and contains functionality for importing image data from file and passing specific image bands forward for subsequent processing. The ImageSequence class holds information about an image sequence, i.e. a collection of CamImage objects, from which specific images and image pairs can be called.

1.4.5 Velocity.py

Calculates velocities and homography. This can either be achieved through the Velocity class for processing velocities and homography through a series of images, or using the functions provided within the script for processing velocities and homography between an image pair.

1.4.6 Area.py

Automated and manual detection of surface areas from imagery (e.g. supraglacial lakes, meltwater plume surface extent). This can either be achieved through the Area class for defining areas of interest through a series of images, or using the functions provided within the script for defining areas of interest in a single image.

1.4.7 Line.py

Manual detection of line features from imagery (e.g. glacier terminus position). This can either be achieved through the Line class for defining line features through a series of images, or using the functions provided within the script for defining line features in a single image.

1.4.8 Utilities.py

This module contains a set of functions for plotting and interpolating data.

CHAPTER

TWO

PYTRX GUIDE

Detailed documentation is included in the 8 scripts that make up PyTrx. Each script contains classes and functions for handling each aspect needed for photogrammetric processing.

For beginners in programming, it is advised to look at the example applications provided in the GitHub repository and adapt them accordingly for your own use. For experienced programmers, get stuck in. Feel free to contact us if you run into major problems or have constructive comments that will help us further PyTrx and its capabilities. We will not respond to minor troubleshooting or unconstructive comments.

2.1 CamEnv

Handles the associated data with the camera environment. The GCPs class handles the Ground Control Points (GCPs) and their correspondence to the associated DEM and CamImage object. The CamCalib class handles information concerning the camera calibration, i.e. the intrinsic camera matrix and lens distortion coefficients. This class contains functionality for reading in calibration files from .txt and .mat formats. The CamEnv compiles all the information about the camera environment from the GCPs and CamCalib classes, and also contains information about the camera object (pose and location). This is also where georectification functionality is held, with functions for projection and inverse projection. The class is initialised using a .txt file containing file path directories to all the associated data files.

2.2 DEM

Handles the DEM data. This currently supports .mat and .tif file types. The ExplicitRaster class represents a DEM as a numeric raster with explicit XY cell referencing in each grid cell. The class includes functions for densification, calculating viewsheds, and incorporates unbound functions that import a DEM file from .mat and .tif formats.

2.3 FileHandler

This module contains a set of functions for reading in data from files (such as image data and calibration information) and writing out data.

2.4 Images

Handles the image data, and the image sequence. The CamImage class holds information about a singular image and contains functionality for importing image data from file and passing specific image bands forward for subsequent processing. The ImageSequence class holds information about an image sequence, i.e. a collection of CamImage objects, from which specific images and image pairs can be called.

2.5 Velocity

Calculates velocities and homography. This can either be achieved through the Velocity class for processing velocities and homography through a series of images, or using the functions provided within the script for processing velocities and homography between an image pair.

2.6 Area

Automated and manual detection of surface areas from imagery (e.g. supraglacial lakes, meltwater plume surface extent). This can either be achieved through the Area class for defining areas of interest through a series of images, or using the functions provided within the script for defining areas of interest in a single image.

2.7 Line

Manual detection of line features from imagery (e.g. glacier terminus position). This can either be achieved through the Line class for defining line features through a series of images, or using the functions provided within the script for defining line features in a single image.

2.8 Utilities

This module contains a set of functions for plotting and interpolating data.

CHAPTER

THREE

PYTRX MODULES

3.1 Area module

The Area module handles the functionality for obtaining areal measurements from oblique time-lapse imagery. Specifically, this module contains functions for: (1) Performing automated and manual detection of areal extents in oblique imagery; and (2) Determining real-world surface areas from oblique imagery.

class Area.**Area**(*imageList*, *cameraenv*, *hmatrix*, *calibFlag=True*, *band='L'*, *equal=True*)

Bases: *Images.ImageSequence*

A class for processing change in area (i.e. a lake or plume) through an image sequence, with methods to calculate extent change in an image plane (px) and real areal change via georectification.

Parameters

- **imageList** (*str/list*) – List of images, for the PyTrx.Images.ImageSequence object
- **cameraenv** (*str*) – Camera environment parameters which can be read into the PyTrx.CamEnv.CamEnv object as a text file
- **hmatrix** (*arr*) – Homography matrix
- **calibFlag** (*bool*) – An indicator of whether images are calibrated, for the PyTrx.Images.ImageSequence object
- **band** (*str, optional*) – String denoting the desired image band, default to 'L' (grayscale)
- **equal** (*bool, optional*) – Flag denoting whether histogram equalisation is applied to images (histogram equalisation is applied if True). Default to True.

calcAutoAreas(*colour=False*, *verify=False*)

Detects areas of interest from a sequence of images, and returns pixel and xyz areas.

Parameters

- **colour** (*bool, optional*) – Flag to denote whether colour range for detection should be defined for each image or only once, default to False
- **verify** (*bool, optional*) – Flag to denote whether detected polygons should be manually verified by user, default to False

Returns XYZ and UV area information

Return type list

calcManualAreas()

Manually define areas of interest in a sequence of images. User input is facilitated through an interactive plot to click around the area of interest

Returns XYZ and UV area information

Return type list

setColourrange(*upper*, *lower*)
Manually define the RBG colour range that will be used to filter the image/images.

Parameters

- **upper** (*int*) – Upper value of colour range
- **lower** (*int*) – Lower value of colour range

setEnhance(*diff*, *phi*, *theta*)
Set image enhancement parameters. Change brightness and contrast of image using phi and theta variables. Change phi and theta values accordingly. See enhanceImg function for detailed explanation of the parameters.

Parameters

- **diff** (*str*) – Inputted as either 'light or 'dark', signifying the intensity of the image pixels. 'light' increases the intensity such that dark pixels become much brighter and bright pixels become slightly brighter. 'dark' decreases the intensity such that dark pixels become much darker and bright pixels become slightly darker.
- **phi** (*int*) – Defines the intensity of all pixel values
- **theta** (*int .*) – Defines the number of "colours" in the image, e.g. 3 signifies that all the pixels will be grouped into one of three pixel values

setMax(*maxMaskPath*, *maxim*)
Set image in sequence which pictures the maximum extent of the area of interest.

Parameters

- **maxMaskPath** (*str*) – File path to mask with maximum extent
- **maxim** (*arr*) – Image with maximum extent

setPXExt(*xmin*, *xmax*, *ymin*, *ymax*)
Set plotting extent. Setting the plot extent will make it easier to define colour ranges and verify areas.

Parameters

- **xmin** (*int*) – X-axis minimum value.
- **xmax** (*int*) – X-axis maximum value.
- **ymin** (*int*) – Y-axis minimum value.
- **ymax** (*int*) – Y-axis maximum value.

setThreshold(*number*)
Set threshold for number of polgons kept from an image.

Parameters **number** (*int*) – Number denoting the number of detected polygons that will be retained

verifyAreas(*areas*, *invprojvars*)
Method to manually verify all polygons in images. Plots sequential images with detected polygons and the user manually verifies them by clicking them.

Parameters

- **area** (*list*) – XYZ and UV area information
- **invprojvars** (*list*) – Inverse projection variables [X,Y,Z,uv0]
- **verified** (*list*) – Verified XYZ and UV area information

Area.**calcAutoArea**(*img*, *imn*, *colourrange*, *hmatrix=None*, *threshold=None*, *invprojvars=None*)

Detects areas of interest from a given image, and returns pixel and xyz areas along with polygon coordinates. Detection is performed from the image using a predefined RBG colour range. The colour range is then used to extract pixels within that range using the OpenCV function inRange. If a threshold has been set (using the setThreshold function) then only nth polygons will be retained. XYZ areas and polygon coordinates are only calculated when a set of inverse projection variables are provided.

Parameters

- **img** (*arr*) – Image array
- **imn** (*str*) – Image name
- **colourrange** (*list*) – RBG colour range for areas to be detected from
- **hmatrix** (*arr*) – Homography matrix, default to None
- **threshold** (*int, optional*) – Threshold number of detected areas to retain, default to None
- **invprojvars** (*list, optional*) – Inverse projection variables [X,Y,Z,uv0], default to None

Returns Four list items containing 1) the sum of total detected areas (xyz), 2) XYZ coordinates of detected areas, 3) Sum of total detected areas (px), and 4) UV coordinates of detected areas

Return type list

Area.**calcManualArea**(*img*, *imn*, *hmatrix=None*, *pxplot=None*, *invprojvars=None*)

Manually define an area in a given image. User input is facilitated through an interactive plot to click around the area of interest. XYZ areas are calculated if a set of inverse projection variables are given.

Parameters

- **img** (*arr*) – Image array
- **imn** (*str*) – Image name
- **hmatrix** (*arr*) – Homography matrix, default to None
- **pxplot** (*list, optional*) – Plotting extent for manual area definition, default to None
- **invprojvars** (*list, optional*) – Inverse projection variables [X,Y,Z,uv0], default to None

Returns Four list items containing 1) the sum of total detected areas (xyz), 2) XYZ coordinates of detected areas, 3) Sum of total detected areas (px), and 4) UV coordinates of detected areas

Return type list

Area.**defineColourrange**(*img*, *imn*, *pxplot=None*)

Define colour range manually by clicking on the lightest and darkest regions of the target extent that will be defined. Plot interaction information: Left click to select, right click to undo selection, close the image window to continue, and the window automatically times out after two clicks.

Parameters

- **img** (*arr*) – Image array
- **imn** (*str*) – Image name
- **pxplot** (*list, optional*) – Plotting extent for manual area definition, default to None

Returns List containing the upper and lower boundary for pixel detection

Return type list

Area.**getOGRArea**(*pts*)
Get real world OGR polygons (.shp) from xyz poly pts with real world points which are compatible with mapping software (e.g. ArcGIS).

Parameters **pts** (*arr*) – UV/XYZ coordinates of a given area shape

Returns List of OGR geometry polygons

Return type list

3.2 CamEnv module

The Camera Environment module contains the object-constructors and functions for: (1) Representing a camera model in three-dimensional space; and (2) Effective translation of measurements in an XY image plane to XYZ real-world coordinates. The projection and inverse transformation functions are based on those available in the ImGRAFT toolbox for Matlab. Translations from ImGRAFT are noted in related script comments.

class CamEnv.**CamCalib**(**args*)
Bases: object

This base class models a standard camera calibration matrix as per OpenCV, MatLab and ImGRAFT. The class uses a standard pinhole camera model, drawing on the functions within OpenCV. A scene view is formed by projecting 3D points into the image plane using a perspective transformation. The camera intrinsic matrix is defined as a 3 x 3 array: [fx,0,0][s,fy,0] [cx,cy,1], where fx and fy is the camera focal length (in pixel units) and cx and cy as the location of the image centre (in pixels too), s is the skew, and cx and cy are the image dimensions in pixels. In addition, the radial distortion and tangential distortion are represented as a series of coefficients. These distortions are introduced by discrepancies in the camera lens and between the lens and the camera sensor: 1) Radial Distortion Coefficients: k ([k1,k2,k3,k4,k5,k6]), between 2 and 6 coefficients needed; and 2) Tangential Distortion Coefficients: p ([p1,p2]) The object can be initiated directly either as a list of three elements for each of the intrinsic, tangential and radial arrays, or by referencing a file (.mat or .txt) containing the calibration data in a pre-designated format.

Parameters **args** (*str*) – Either a calibration text file, a series of calibration text files, a list of raw parameters, or a set of calibration images (along with calibration chessboard dimensions)

checkMatrix(*matrix*)
Function to support the calibrate function. Checks and converts the intrinsic matrix to the correct format for calibration with opencv.

Parameters **matrix** (*arr*) – Intrinsic camera matrix

Returns The object's intrinsic matrix (checked), tangential distortion and radial distortion information

Return type list

getCalibdata()
Return camera matrix, and tangential and radial distortion coefficients.

getCamMatrix()
Return camera matrix.

getCamMatrixCV2()
Return camera matrix in a structure that is compatible with subsequent photogrammetric processing using OpenCV.

getDistortCoeffsCV2()
Return radial and tangential distortion coefficients.

reportCalibData()
Self reporter for Camera Calibration object data.

class CamEnv.**CamEnv**(*envFile*)
Bases: *CamEnv.CamCalib*

A class to represent the camera object, containing the intrinsic matrix, distortion parameters and camera pose (position and direction). Also inherits from the PyTrx.CamEnv.CamCalib object, representing the intrinsic camera information. This object can be initialised either through an environment file (and passed to the initialiser as a filepath), or with the set intput parameters

Parameters

- **name** (*str*) – The reference name for the camera
- **GCPpath** (*str*) – The file path of the GCPs, for the GCPs object
- **DEMpath** (*str*) – The file path for the DEM, for the GCPs object
- **imagePath** (*str*) – The file path for the GCP reference image, for the GCPs object
- **calibPath** (*str*) – The file path for the calibration file. This can be either as a .mat Matlab file or a text file. The text file should be of the following tab delimited format: RadialDistortion [k1 k2 k3…k7], TangentialDistortion [p1 p2], IntrinsicMatrix [x y z][x y z][x y z], End
- **coords** (*list*) – The x,y,z coordinates of the camera location, as a list
- **ypr** (*list*) – The yaw, pitch and roll of the camera, as a list

dataFromFile(*filename*)
Read CamEnv data from .txt file containing keywords and filepaths to associated data.

Parameters **filename** (*str*) – Environment file path

Returns Camera environment information (name, GCP filepath, DEM filepath, image filepath, calibration file path, camera coordinates, camera pose (ypr) and DEM densification factor)

Return type list

getDEM()
Return DEM as PyTrx.DEM.ExplicitRaster object.

Returns DEM object

Return type PyTrx.DEM.ExplicitRaster

getRefImageSize()
Return the dimensions of the reference image.

Returns Image size

Return type arr

optimiseCamEnv(*optimise*, *optmethod='trf'*, *show=False*)
Optimise projection variables in the camera environment. The precise parameters to optimise are defined by the optimise variable.

Parameters

- **optimise** (*str*) – Parameters to optimise - 'YPR' (optimise camera pose only), 'EXT' (optimise external camera parameters), 'INT' (optimise internal camera parameters), or 'ALL' (optimise all projection parameters)
- **optmethod** (*str, optional*) – Optimisation method, default to 'trf'

- **show** (*bool, optional*) – Flag to denote if optimisation output should be plotted, default to False

reportCamData()
Reporter for testing that the relevant data has been successfully imported. Testing for camera Environment name, camera location (xyz), reference image, DEM, DEM densification, GCPs, yaw pitch roll, camera matrix, and distortion coefficients.

showCalib()
Plot corrected and uncorrected reference image.

showGCPs()
Plot GCPs in image plane and DEM scene.

showPrincipalPoint()
Plot Principal Point on reference image.

showResiduals()
Show positions of xyz GCPs and projected GCPs, and residual differences between their positions. This can be used as a measure of a error in the georectification of measurements.

class CamEnv.**GCPs**(*dem*, *GCPpath*, *imagePath*)
Bases: object

A class representing the geography of the camera scene. Contains ground control points, as the world and image points, the DEM data and extent, and the image the ground control points correspond to, as an Image object.

Parameters

- **dem** (*str*) – The file path of the ASCII DEM
- **GCPpath** (*str*) – The file path of the GCP text file, with a header line, and tab delimited x, y, z world coordinates and xy image on each line
- **imagePath** (*str*) – The file path of the image the GCPs correspond to

getDEM()
Return the dem object.

getGCPs()
Return the world and image GCPs.

getImage()
Return the GCP reference image.

CamEnv.**calibrateImages**(*imageFiles*, *xy*, *refine=None*)
Function for calibrating a camera from a set of input calibration images. Calibration is performed using OpenCV's chessboard calibration functions. Input images (imageFile) need to be of a chessboard with regular dimensions and a known number of corner features (xy). Please note that OpenCV's calibrateCamera function is incompatible between different versions of OpenCV. Included here is the function for version 3. Please see OpenCV's documentation for older versions.

Parameters

- **imageFiles** (*list*) – List of image file names
- **xy** (*list*) – Chessboard corner dimensions [rows, columns]
- **refine** (*int, optional*) – OpenCV camera model refinement method - cv2.CALIB_FIX_PRINCIPAL_POINT (fix principal point), cv2.CALIB_FIX_ASPECT_RATIO (Fix aspect ratio), cv2.CALIB_FIX_FOCAL_LENGTH (Fix focal length), cv2.CALIB_FIX_INTRINSIC (Fix camera model), cv2.CALIB_FIX_K1...6 (Fix radial coefficient

1-6), cv2.CALIB_FIX_TANGENT_DIST (Fix tangential coefficients), cv2.CALIB_USE_INTRINSIC_GUESS (Use initial intrinsic values), cv2.CALIB_ZERO_TANGENT_DIST (Set tangential distortion coefficients to zero), cv2.CALIB_RATIONAL_MODEL (Calculate radial distortion coefficients k4, k5, and k6). Default to None

Returns A list containing the camera intrinsic matrix (arr), and tangential (arr) and radial distortion coefficents (arr), and the Camera calibration error (int)

Return type arr/int

CamEnv.**computeResidualsUV**(*params*, *stable*, *GCPxyz*, *GCPuv*, *refimg*, *optimise='YPR'*)

Function for computing the pixel difference between GCP image points and GCP projected XYZ points. This function is used in the optimisation function (optimiseCamera), with parameters for optimising defined in the first variable and stable parameters defined in the second. If no optimisable parameters are given and the optimise flag is set to None then residuals are computed for the original parameters (i.e. no optimisation).

Parameters

- **params** (*arr*) – Optimisable parameters, given as a 1-D array of shape (m,)
- **stable** (*list*) – Stable parameters that will not be optimised
- **GCPxyz** (*arr*) – GCPs in scene space (x,y,z)
- **GCPuv** (*arr*) – GCPs in image space (u,v)
- **refimg** (PyTrx.Images.CamImage/str/arr) – Reference image, given as a CamImage object, file path string, or image array
- **optimise** (*str*) – Flag denoting which variables will be optimised: YPR (camera pose only), INT (internal camera parameters), EXT (external camera parameters), LOC (all parameters except camera location), or ALL (all projection parameters)

Returns Pixel difference between UV and projected XYZ position of each GCP

Return type arr

CamEnv.**computeResidualsXYZ**(*invprojvars*, *GCPxyz*, *GCPuv*, *dem*)

Function for computing the pixel difference between GCP image points and GCP projected XYZ points. This function is used in the optimisation function (optimiseCamera), with parameters for optimising defined in the first variable and stable parameters defined in the second. If no optimisable parameters are given and the optimise flag is set to None then residuals are computed for the original parameters (i.e. no optimisation).

Parameters

- **params** (*arr*) – Optimisable parameters, given as a 1-D array of shape (m,)
- **stable** (*list*) – Stable parameters that will not be optimised
- **GCPxyz** (*arr*) – GCPs in scene space (x,y,z)
- **GCPuv** (*arr*) – GCPs in image space (u,v)
- **refimg** (PyTrx.Images.CamImage/str/arr) – Reference image, given as a CamImage object, file path string, or image array
- **optimise** (*str*) – Flag denoting which variables will be optimised: YPR (camera pose only), INT (internal camera parameters), EXT (external camera parameters), LOC (all parameters except camera location), or ALL (all projection parameters)

Returns Array denoting pixel difference between UV and projected XYZ position of each GCP

Return type arr

CamEnv.**constructDEM**(*dempath*, *densefactor*)
Construct DEM from a given file path and densification factor.

Parameters

- **dempath** (*str*) – DEM filepath
- **densefactor** (*int*) – Densification factor

CamEnv.**getRotation**(*camDirection*)
Calculates camera rotation matrix calculated from view direction.

Parameters **camDirection** (*arr*) – Camera pose (yaw,pitch,roll)

Returns Rotation matrix as array

Return type arr

CamEnv.**optimiseCamera**(*optimise*, *projvars*, *GCPxyz*, *GCPuv*, *optmethod='trf'*, *show=False*)
Optimise camera parameters using the pixel differences between a set of image GCPs and projected XYZ GCPs. The optimisation routine adopts the least_square function in scipy's optimize tools, using either the Trust Region Reflective algorithm, the dogleg algorithm or the Levenberg-Marquardt algorithm to refine a set group of projection parameters - camera pose only, the internal camera parameters (i.e. radial distortion, tangential distortion, focal length, principal point), the external camera parameters (i.e. camera location, camera pose), or all projection parameters (i.e. camera location, camera pose, radial distortion, tangential distortion, focal length, principal point). The Trust Region Reflective algorithm is generally a robust method, ideal for solving many variables (default). The Dogleg algorithm is ideal for solving few variables. The Levenberg-Margquardt algorithm is the most efficient method, ideal for solving few variables. Pixel differences between a set of image GCPs and projected XYZ GCPs are calculated and refined within the optimisation function, performing iterations until an optimum solution is reached. A new set of optimised projection parameters are returned.

Parameters

- **optimise** (*str*) – Flag denoting which variables will be optimised: YPR (camera pose only), INT (internal camera parameters), EXT (external camera parameters), LOC (all parameters except camera location), or ALL (all projection parameters)
- **projvars** – Projection parameters [camera location, camera pose, radial distortion, tangential distortion, focal length, principal point, reference image]
- **GCPuv** (*arr*) – UV positions for GCPs, as shape (m, 2)
- **optmethod** (*str*) – Optimisation method: 'trf' (Trust Region Reflective algorithm), 'dogbox' (dogleg algorithm), or 'lm' (Levenberg-Marquardt algorithm)
- **show** (*bool .*) – Flag denoting whether plot of residuals should be shown

Typeprojvars list

Returns A list containing the optimised projection parameters. If optimisation fails then None is returned

Return type list

CamEnv.**projectUV**(*uv*, *invprojvars*)
Inverse project image coordinates (uv) to xyz world coordinates using inverse projection variables (set using setProjection function). This function is primarily adopted from the ImGRAFT projection function found in camera.m: uv,depth,inframe=cam.project(xyz)

Parameters

- **uv** (*arr*) – Pixel coordinates in image
- **invprojvars** (*list*) – Inverse projection variables [X,Y,Z,uv0]

Returns World coordinates

Return type arr

CamEnv.**projectXYZ**(*camloc*, *camdirection*, *radial*, *tangen*, *foclen*, *camcen*, *refimg*, *xyz*)
Project the xyz world coordinates into the corresponding image coordinates (uv). This is primarily executed using the ImGRAFT projection function found in camera.m: uv,depth,inframe=cam.project(xyz)

Parameters

- **camloc** (*arr*) – Camera location [X,Y,Z]
- **camdirection** (*arr*) – Camera pose (yaw, pitch, roll)
- **radial** (*arr*) – Radial distortion coefficients
- **tangen** (*arr*) – Tangential distortion coefficients
- **foclen** (*arr*) – Camera focal length
- **camcen** (*arr*) – Camera principal point
- **refimg** (*arr*) – Reference image (function only uses the image dimensions)
- **xyz** (*arr*) – world coordinates

Returns Pixel coordinates in image (arr), view depth (int), and a Boolean vector containing whether each projected 3D point is inside the frame

Return type arr/int

CamEnv.**setProjection**(*dem*, *camloc*, *camdir*, *radial*, *tangen*, *foclen*, *camcen*, *refimg*, *viewshed=True*)
Set the inverse projection variables.

Parameters

- **dem** (PyTrx.DEM.ExplicitRaster) – DEM object
- **camloc** (*arr*) – Camera location (X,Y,Z)
- **camdir** (*arr*) – Camera pose [yaw, pitch, roll]
- **radial** (*arr*) – Radial distortion coefficients
- **tangen** (*arr*) – Tangential distortion coefficients
- **foclen** (*arr*) – Camera focal length
- **camcen** (*arr*) – Camera principal point
- **refimg** (*arr*) – Reference image (function only uses the image dimensions)
- **viewshed** (*bool*) – Flag to denote if viewshed from camera should be determined before projection

Returns Inverse projection coefficients [X,Y,Z,uv0]

Return type list

3.3 DEM module

The DEM module contains functionality for handling DEM data and implementing this data into the PyTrx.CamEnv.CamEnv object class.

DEM.**DEM_FromMat**(*matfile*)
Function for loading a DEM array from a Matlab (.mat) file containing separate X, Y, Z matrices.

Parameters **matfile** (*str*) – DEM .mat filepath

Returns A DEM object

Return type PyTrx.DEM.ExplicitRaster

DEM.**DEM_FromTiff**(*tiffFile*)
Function for loading a DEM array from a .tiff file containing raster-formatted data. The tiff data importing is handled by GDAL.

Parameters **tiffFile** (*str*) – DEM .tif filepath

Returns A DEM object

Return type PyTrx.DEM.ExplicitRaster

class DEM.**ExplicitRaster**(*X*, *Y*, *Z*, *nodata=nan*)
Bases: object

A class to represent a numeric Raster with explicit XY cell referencing in each grid cell.

Parameters

- **X** (*arr*) – X data
- **Y** (*arr*) – Y data
- **Z** (*arr*) – Z data
- **nodata** (*int, optional*) – Condition for NaN data values, default to 'nan'

densify(*densefac=2*)
Function to densify the DEM array by a given densification factor. The array is multiplied by the given densification factor and then subsequently values are interpolated using the SciPy function RectBivariateSpline. The densification factor is set to 2 by default, meaning that the size of the DEM array is doubled.

Parameters **densefac** (*int*) – Densification factor

Returns Densified DEM

Return type PyTrx.DEM.ExplicitRaster

getCols()
Return the number of columns in the DEM data array.

Returns DEM column count

Return type int

getData(*dim=None*)
Return DEM data. XYZ dimensions can be individually called with the dim input variable (integer: 0, 1, or 2).

Parameters **dim** (*int*) – Dimension to retrieve (0, 1, or 2), default to None

Returns DEM dimension as array

Return type arr

getExtent()
Return DEM extent.

Returns DEM extent

Return type list

getNoData()
Return fill value for no data in DEM array.

Returns DEM nan fill value

Return type int

getRows()
Return the number of rows in the DEM data array.

Returns DEM row count

Return type int

getShape()
Return the shape of the DEM data array.

Returns DEM shape

Return type arr

getZ()
Return height (Z) data of DEM.

Returns DEM Z values

Return type arr

getZcoord(*x*, *y*)
Return height (Z) at a given XY coordinate in DEM.

Parameters

- **x** (*int*) – X coordinate
- **y** (*int*) – Y coordinate

Returns DEM Z value for given coordinate

Return type int

reportDEM()
Self reporter for DEM class object. Returns the number of rows and columns in the array, how NaN values in the array are filled, and the data extent coordinates.

subset(*cmin*, *cmax*, *rmin*, *rmax*)
Return a specified subset of the DEM array.

Parameters

- **cmin** (*int*) – Column minimum extent
- **cmax** (*int*) – Column maximum extent
- **rmin** (*int*) – Row minimum extent
- **rmax** (*int*) – Row maximum extent

Returns Subset of DEM

Return type PyTrx.DEM.ExplicitRaster

DEM.**load_DEM**(*demfile*)
Function for loading DEM data from different file types, which is automatically detected. Recognised file types: .mat and .tif.

Parameters **demfile** (*str*) – DEM filepath

Returns A DEM object

Return type PyTrx.DEM.ExplicitRaster

DEM.**voxelviewshed**(*dem*, *viewpoint*)
Calculate a viewshed over a DEM from a given viewpoint in the DEM scene. This function is based on the viewshed function (voxelviewshed.m) available in ImGRAFT. The ImGRAFT voxelviewshed.m script is available at: http://github.com/grinsted/ImGRAFT/blob/master/voxelviewshed.m

Parameters

- **dem** (PyTrx.DEM.ExplicitRaster) – A DEM object
- **viewpoint** (*list*) – 3-element vector specifying the viewpoint

Returns Boolean visibility matrix (which is the same size as dem)

Return type arr

3.4 FileHandler module

Th FileHandler module contains all the functions called by a PyTrx object to load and export data.

FileHandler.**checkMatrix**(*matrix*)
Function to support the calibrate function. Checks and converts the intrinsic matrix to the correct format for calibration with OpenCV.

Parameters **matrix** (*arr*) – Inputted matrix for checking

Returns Validated matrix

Return type arr

FileHandler.**importAreaData**(*xyzfile*, *pxfile*)
Import xyz and px data from text files.

Parameters

- **xyzfile** (*str*) – File directory to xyz coordinates
- **pxfile** (*str*) – File directory to uv coordinates

Returns Coordinates and areas of detected areas

Return type list

FileHandler.**importAreaFile**(*fname*, *dimension*)
Import pixel polygon data from text file and compute pixel extents.

Parameters

- **fname** (*str*) – Path to the text file containing the UV coordinate data
- **dimension** (*int*) – Integer denoting the number of dimensions in coordinates

Returns Two lists, containing the UV coordinates for polygons and the pixel areas for polygons

Return type list

FileHandler.**importLineData**(*xyzfile*, *pxfile*)
Import xyz and px data from text files.

Parameters

- **xyzfile** (*str*) – File directory to xyz coordinates
- **pxfile** (*str*) – File directory to uv coordinates

Returns Coordinates and lengths of detected lines

Return type list

FileHandler.**importLineFile**(*fname*, *dimension*)
Import XYZ line data from text file and compute line lengths.

Parameters

- **fname** (*str*) – Path to the text file containing the XYZ coordinate data
- **dimension** (*int*) – Number of dimensions in point coordinates i.e. 2 or 3

Returns List containing line coordinates and lengths

Return type list

FileHandler.**lineSearch**(*lineList*, *search*)
Function to supplement the readCalib function. Given an input parameter to search within the file, this will return the line numbers of the data.

Parameters

- **lineList** (*list*) – List of strings within a file line
- **search** (*str*) – Target keyword to search for

Returns Line numbers with keyword match

Return type list

FileHandler.**readCalib**(*fileName*, *paramList*)
Function to find camera calibrations from a file given a list or Matlab file containing the required parameters. Returns the parameters as a dictionary object. Compatible file structures: 1) .txt file ("RadialDistortion [k1,k2,k3…k8], TangentialDistortion [p1,p2], IntrinsicMatrix [fx 0. 0.][s fy 0.][cx cy 1] End"); 2) .mat file (Camera calibration file output from the Matlab Camera Calibration App (available in the Computer Vision Systems toolbox).

Parameters

- **fileName** (*str*) – File directory for calibration file
- **paramList** (*list*) – List of strings denoting keywords to look for in calibration file

Returns Calibration parameters denoted by keywords

Return type list

FileHandler.**readGCPs**(*fileName*)
Function to read ground control points from a .txt file. The data in the file is referenced to under a header line. Data is appended by skipping the header line and finding the world and image coordinates from each line.

Parameters **fileName** (*str*) – File path directory for GCP file

Returns Two arrays containing the GCPs in xyz coordinates, and the GCPs in image coordinates

Return type array

FileHandler.**readImg**(*path*, *band='L'*, *equal=True*)

Function to prepare an image by opening, equalising, converting to either grayscale or a specified band, then returning a copy.

Parameters

- **path** (*str*) – Image file path directory
- **band** (*str*) – Desired band output - 'R': red band; 'B': blue band; 'G': green band; 'L': grayscale (default).
- **equal** (*bool*) – Flag to denote if histogram equalisation should be applied, defaults to True

Returns Image array

Return type arr

FileHandler.**readMask**(*img*, *writeMask=None*)

Function to create a mask for point seeding using PIL to rasterize polygon. The mask is manually defined by the user using the pyplot ginput function. This subsequently returns the manually defined area as a .jpg mask. The writeMask file path is used to either open the existing mask at that path or to write the generated mask to this path.

Parameters

- **img** (*arr*) – Image to define mask in
- **writeMask** (*str, optional*) – File destination that mask output is written to, default to None

Returns Array defining the image mask

Return type arr

FileHandler.**readMatrixDistortion**(*path*)

Function to support the calibrate function. Returns the intrinsic matrix and distortion parameters required for calibration from a given file.

Parameters **path** (*str*) – Directory of calibration file

Returns Three camera matrix parameters: 1) the intrinsic matrix as a 3x3 array, including focal length, principal point, and skew (arr); 2) Tangential distortion values (p1, p2) (arr); and 3) Radial distortion values (k1, k2… k6)

Return type arr

FileHandler.**returnData**(*lines*, *data*)

Function to supplement the importCalibration function. Given the line numbers of the parameter data (the ouput of the lineSearch function), this will return the data.

Parameters

- **lines** (*list*) – Given line numbers to extract data from
- **data** (*list*) – Raw line data

Returns Extracted data

Return type arr

FileHandler.**writeAreaCoords**(*pxpts*, *xyzpts*, *imn*, *pxdestination*, *xyzdestination*)

Write UV and XYZ area coordinates to text files. These file types are compatible with the importing tools (importAreaPX, importAreaXYZ).

Parameters

- **xyzarea** (*list*) – XYZ areas
- **xyzpts** (*list*) – XYZ coordinates
- **imn** (*list*) – Image names
- **pxdestination** (*str*) – File directory where UV coordinates will be written to
- **xyzdestination** (*str*) – File directory where XYZ coordinates will be written to

FileHandler.**writeAreaFile**(*pxareas*, *xyzareas*, *imn*, *destination*)
Write UV and XYZ areas to csv file.

Parameters

- **pxarea** (*list*) – Pixel extents
- **xyzarea** (*list*) – XYZ areas
- **imn** (*list*) – Image names
- **destination** (*str*) – File directory where csv file will be written to

FileHandler.**writeAreaSHP**(*xyzpts*, *imn*, *fileDirectory*, *projection=None*)
Write OGR real polygon areas (from ALL images) to file in a .shp file type that is compatible with ESRI mapping software.

Parameters

- **xyzpts** (*list*) – XYZ coordinates for polygons
- **imn** (*list*) – Image name
- **fileDirectory** (*str*) – Destination that shapefiles will be written to
- **projection** (*int/str*) – Coordinate projection that the shapefile will exist in. This can either be an ESPG number (expressed as an integer) or a well-known geographical coordinate system (expressed as a string). Well-known geographical coordinate systems are: 'WGS84', 'WGS72', NAD83' or 'EPSG:n'

FileHandler.**writeCalibFile**(*intrMat*, *tanDis*, *radDis*, *fname*)
Write camera calibration data to .txt file, including camera matrix, and radial and tangential distortion parameters.

Parameters

- **intrMat** (*arr*) – Intrinsic camera matrix
- **tanDis** (*arr*) – Tangential distortion parameters
- **radDis** (*arr*) – Radial distortion parameters
- **fname** (*str*) – Directory to write file to

FileHandler.**writeHomogFile**(*homog*, *imn*, *fname*)
Function to write all homography data from a given timeLapse sequence to .csv file. Data is formatted as sequential columns containing the following information: Image pair 1 name, Image pair 2 name, Homography matrix (i.e. all values in the 3x3 matrix), Number of features tracked, X mean displacement, Y mean displacement, X standard deviation, Y standard deviation, Mean error magnitude, Mean homographic displacement, and Signal-to-noise ratio.

Parameters

- **homog** (*list*) – Homography [mtx, im0pts, im1pts, ptserr, homogerr]
- **imn** (*list*) – List of image names
- **fname** (*str*) – Directory for file to be written to

FileHandler.**writeLineCoords**(*uvpts*, *xyzpts*, *imn*, *pxdestination*, *xyzdestination*)

Write UV and XYZ line coordinates to text file. These file types are compatible with the importing tools (importLinePX, importLineXYZ)

Parameters

- **uvpts** (*list*) – Pixel coordinates
- **xyzpts** (*list*) – XYZ coordinates
- **imn** (*list*) – Image names
- **pxdestination** (*str*) – File directory where UV coordinates will be written to
- **xyzdestination** (*str*) – File directory where XYZ coordinates will be written to

FileHandler.**writeLineFile**(*pxline*, *xyzline*, *imn*, *destination*)

Write UV and XYZ line lengths to csv file.

Parameters

- **pxline** (*list*) – Pixel line lengths
- **xyzline** (*list*) – XYZ line lengths
- **imn** (*list*) – Image names
- **destination** (*str*) – File directory where output will be written to

FileHandler.**writeLineSHP**(*xyzpts*, *imn*, *fileDirectory*, *projection=None*)

Write OGR real line features (from ALL images) to file in a .shp file type that is compatible with ESRI mapping software.

Parameters

- **xyzpts** (*list*) – XYZ coordinates for polygons
- **imn** (*list*) – Image name
- **fileDirectory** (*str*) – Destination that shapefiles will be written to
- **projection** (*int/str*) – Coordinate projection that the shapefile will exist in. This can either be an ESPG number (expressed as an integer) or a well-known geographical coordinate system (expressed as a string). Well-known geographical coordinate systems are: 'WGS84', 'WGS72', NAD83' or 'EPSG:n'

FileHandler.**writeVeloFile**(*xyzvel*, *uvvel*, *homog*, *imn*, *fname*)

Function to write all velocity data from a given timeLapse sequence to .csv file. Data is formatted as sequential columns containing the following information: Image pair 1 name, Image pair 2 name, Average xyz velocity, Number of features tracked, Average pixel velocity, Homography residual mean error (RMS), and Signal-to-noise ratio.

Parameters

- **xyzvel** (*list*) – XYZ velocities
- **uvvel** (*list*) – Pixel velocities
- **homog** (*list*) – Homography [mtx, im0pts, im1pts, ptserr, homogerr]
- **imn** – List of image names
- **imn** – list
- **fname** (*str*) – Filename for output file. File destination can also specified

FileHandler.**writeVeloSHP**(*xyzvel*, *xyzerr*, *xyz0*, *imn*, *fileDirectory*, *projection=None*)

Write OGR real velocity points (from ALL images) to file in a .shp file type that is compatible with ESRI mapping software.

Parameters

- **xyzvel** (*list*) – XYZ velocities
- **xyz0** (*list*) – XYZ pt0
- **imn** (*list*) – Image name
- **fileDirectory** (*str*) – Destination that shapefiles will be written to
- **projection** (*int/str*) – Coordinate projection that the shapefile will exist in. This can either be an ESPG number (expressed as an integer) or a well-known geographical coordinate system (expressed as a string). Well-known geographical coordinate systems are: 'WGS84', 'WGS72', NAD83' or 'EPSG:n'

3.5 Images module

The Images module contains the object-constructors and functions for: (1) Importing and handling image data, specifically RBG, one-band (R, B or G), and grayscale images; and (2) Handling image sequences (i.e. a set of multiple images).

class Images.**CamImage**(*imagePath*, *band='l'*, *equal=True*)

Bases: object

A class representing a raw single band (optical RGB or greyscale). This CamImage object is used in subsequent timelapse analysis. The object contains the image data, image path, image dimensions and timestamp (derived from the image Exif data, if available). Optionally the user can specify whether the red, blue or green values should be used, or whether the images should be converted to grey scale which is the default. No image calibration is undertaken at this point. The default grayscale band option ('l') applies an equalization filter on the image whereas the RGB splits are raw RGB. This could be modified to permit more sophisticated settings of RGB combinations and/or filters with file reading.

Parameters

- **imagePath** (*str*) – The file path to a given image
- **band** (*str*) – Specified image band to pass forward: 'r' - red band; 'b' - blue band; 'g' - green band; and 'l' - grayscale. Default to 'l'
- **equal** (*bool, optional*) – Flag denoting whether histogram equalisation is applied to images (histogram equalisation is applied if True). Default to True

changeBand(*band*)

Change the band you want the image to represent ('r', 'b', 'g' or 'l')

Parameters **band** (*str*) – Image band ('r', 'b', 'g', or 'l')

clearAll()

Clear memory of all retained data.

clearImage()

Clear memory of image data.

clearImageArray()

Clear memory of image array data.

getExif()
Return the exif image size and time stamp data from the image. Image size is returned as a string (height, width). The time stamp is returned as a Python datetime object.

getImage()
Return the image.

getImageArray()
Return the image as an array.

getImageCorr(*cameraMatrix*, *distortP*)
Return the image array that is corrected for the specificied camera matrix and distortion parameters.

Parameters

- **cameraMatrix** (*arr*) – Intrinsic camera matrix
- **distortP** (*arr*) – Lens distortion parameters

Returns Image corrected for image distortion (arr)

Return type arr

getImageEnhance(*diff*, *phi*, *theta*)
Return enhanced image using `PyTrx.Images.enhanceImg()` function.

Parameters

- **diff** (*str*) – Inputted as either 'light or 'dark', signifying the intensity of the image pixels. 'light' increases the intensity such that dark pixels become much brighter and bright pixels become slightly brighter. 'dark' decreases the intensity such that dark pixels become much darker and bright pixels become slightly darker.
- **phi** (*int*) – Defines the intensity of all pixel values
- **theta** (*int*) – Defines the number of "colours" in the image, e.g. 3 signifies that all the pixels will be grouped into one of three pixel values

Returns Enhanced image

Return type arr

getImageName()
Return image name.

getImagePath()
Return the file path of the image.

getImageSize()
Return the size of the image (which is obtained from the image Exif information).

getImageTime()
Return the time of the image (which is obtained from the image Exif information).

getImageType()
Return the image file type.

imageGood()
Return image file path status.

reportCamImageData()
Report image data (file path, image size and datetime).

class Images.**ImageSequence**(*imageList*, *band='L'*, *equal=True*)
Bases: `object`

A class to model a raw collection of CamImage objects, which can subsequently be used for making photogrammetric measurements from.

Parameters

- **imageList** (*str*) – The list of images, which can be passed in 3 ways: 1) As a list of `PyTrx.Image.CamImage` objects; 2) As a list of image paths; and 3) As a folder containing images
- **band** (*str, optional*) – Image band ('r', 'b', 'g', or 'l'), default to 'l'
- **equal** (*bool, optional*) – Flag denoting whether histogram equalisation is applied to images (histogram equalisation is applied if True). Default to True

getImageArrNo(*i*)
Get image array i from image sequence

Parameters **i** (*int*) – Image number in sequence

Returns Image array

Return type arr

getImageFileList()
Return list of image file paths.

getImageNames()
Return list of image file names.

getImageObj(*i*)
Get `PyTrx.Images.CamImage` object i from image sequence

Parameters **i** (*int*) – Image number in sequence

Returns `PyTrx.Images.CamImage`

Return type arr

getImages()
Return image set (i.e. a sequence of CamImage objects).

getLength()
Return length of image set.

`Images.`**enhanceImage**(*img*, *diff*, *phi*, *theta*)
Change brightness and contrast of image using phi and theta variables. Change phi and theta values accordingly.

Parameters

- **img** (*arr*) – Input image array for enhancement
- **diff** (*str*) – Inputted as either 'light or 'dark', signifying the intensity of the image pixels. 'light' increases the intensity such that dark pixels become much brighter and bright pixels become slightly brighter. 'dark' decreases the intensity such that dark pixels become much darker and bright pixels become slightly darker.
- **phi** (*int*) – Defines the intensity of all pixel values
- **theta** (*int .*) – Defines the number of "colours" in the image, e.g. 3 signifies that all the pixels will be grouped into one of three pixel values

Returns Enhanced image.

Return type arr

3.6 Line module

The Line module handles the functionality for obtaining line measurements from oblique time-lapse imagery. Specifically, this module contains functions for: (1) Performing manual detection of lines in oblique imagery; and (2) Determining real-world distances from oblique imagery.

class Line.**Line**(*imageList*, *cameraenv*, *hmatrix*, *calibFlag=True*, *band='L'*, *equal=True*)

Bases: *Images.ImageSequence*

A class for handling lines/distances (e.g. glacier terminus position) through an image sequence, with methods to manually define pixel lines in the image plane and georectify them to generate real-world coordinates and distances. The Line class object primarily inherits from the Area class.

Parameters

- **imageList** (*str/list*) – List of images to be inputted into the PyTrx.Images.ImageSequence object
- **cameraenv** (*str*) – Camera environment parameters which can be read into the PyTrx.CamEnv.CamEnv object as a text file
- **hmatrix** (*arr*) – Homography matrix
- **calibFlag** (*bool, optional .*) – An indicator of whether images are calibrated, for the PyTrx.Images.ImageSequence object, default to True
- **band** (*str, optional .*) – String denoting the desired image band, default to 'L' (grayscale)
- **equal** (*bool, optional*) – Flag denoting whether histogram equalisation is applied to images (histogram equalisation is applied if True). Default to True.

calcManualLines()

Method to manually define pixel lines from an image sequence. The lines are manually defined by the user on an image plot. Returns the line pixel coordinates and pixel length.

Returns XYZ and UV line lengths and coordinates

Return type list

Line.**calcManualLine**(*img*, *imn*, *hmatrix=None*, *invprojvars=None*)

Manually define a line in a given image to produce XYZ and UV line length and corresponding coordinates. Lines are defined through user input by clicking in the interactive image plot. This primarily operates via the pyplot.ginput function which allows users to define coordinates through plot interaction. If inverse projection variables are given, XYZ lines and coordinates are also calculated.

Parameters

- **img** (*arr*) – Image array for plotting.
- **imn** (*str*) – Image name
- **hmatrix** (*arr, optional*) – Homography matrix, default to None
- **invprojvars** (*list, optional*) – Inverse projection variables [X,Y,Z,uv0], default to None

Returns Four list elements containing: line length in xyz (list), xyz coordinates of lines (list), line length in pixels (list), and uvcoordinates of lines (list)

Return type list

Line.**getOGRLine**(*pts*)

Function to construct an OGR line from a set of uv coordinates.

Parameters **pts** (*arr*) – A series of uv coordinates denoting a line

Returns A line object (ogr.Geometry) constructed from the input coordinates

Return type ogr.Geometry

3.7 Utilities module

The Utilities module contains stand-alone functions needed for simple plotting and interpolation. These merely serve as examples and it is highly encouraged to adapt these functions for visualising datasets.

Utilities.**arrowplot**(*xst*, *yst*, *xend*, *yend*, *scale=1.0*, *headangle=15*, *headscale=0.2*)

Plot arrows to denote the direction and magnitude of the displacement. Direction is indicated by the bearing of the arrow, and the magnitude is indicated by the length of the arrow.

Parameters

- **x0** (*arr*) – X coordinates for pt0
- **y0** (*arr*) – Y coordinates for pt0
- **x1** (*arr*) – X coordinates for pt1
- **y1** (*arr*) – Y coordinates for pt1
- **scale** (*int, optional*) – Arrow scale, defaults to 1.0
- **headangle** (*int, optional*) – Plotting angle, defaults to 15
- **headscale** (*int, optional*) – Arrow head scale, defaults to 0.2

Returns Arrow plots as two arrays denoting x and y coordinates

Return type arr

Utilities.**interpolateHelper**(*xyzvel*, *xyz0*, *xyz1*, *method='linear'*)

Function to interpolate a point dataset. This uses functions of the SciPy package to set up a grid (grid) and then interpolate using a linear interpolation method (griddata). Methods are those compatible with SciPy's interpolate.griddata function: 'nearest', 'cubic' and 'linear'.

Parameters

- **xxyzvel** – Input xyz velocities
- **xyz0** (*arr*) – Coordinates (x,y) for points in first image
- **xyz1** (*arr*) – Coordinates (x,y) for points in second image
- **method** (*str, optional*) – Interpolation method, defaults to 'linear'

Returns An interpolated grid of points and grid extent

Return type list

Utilities.**plotAreaPX**(*uv*, *img*, *show=True*, *save=None*)

Plot figure with image overlayed with pixel area features.

Parameters

- **uv** (*arr*) – Input uv coordinates for plotting over image
- **img** (*arr*) – Image array
- **show** (*bool, optional*) – Flag to denote whether the figure is shown, defatuls to True
- **save** (*str, optional*) – Destination file to save figure to, defaults to None

Returns A figure with plotted area measurements overlaid onto a given image

Utilities.**plotAreaXYZ**(*xyz*, *dem*, *show=True*, *save=None*)
Plot figure with image overlayed with xyz coordinates representing either areas or line features.

Parameters

- **xyz** (*arr*) – Input xyz coordinates for plotting
- **dem** (PyTrx.DEM.ExplicitRaster) – Underlying DEM for plotting over
- **show** (*bool, optional*) – Flag to denote whether the figure is shown, defatuls to True
- **save** (*str, optional*) – Destination file to save figure to, defaults to None

Returns A figure with plotted areas overlaid onto a given DEM

Utilities.**plotCalib**(*matrix*, *distortion*, *img*, *imn*)
Function to show camera calibration. Two images are plotted, the first with the original input image and the second with the calibrated image. This calibrated image is corrected for distortion using the distortion parameters held in the PyTrx.CamEnv.CamCalib object.

Parameters

- **matrix** (*arr*) – Camera matrix
- **distortion** (*arr*) – Distortion cofficients
- **img** (*arr*) – Image array
- **imn** (*str*) – Image name

Returns A figure of an uncorred image and corrected image

Utilities.**plotGCPs**(*gcps*, *img*, *imn*, *dem*, *camloc*, *extent=None*)
Function to show the ground control points, on the image and the DEM.

Parameters

- **gcps** (*arr*) – GCPs
- **img** (*arr*) – Image array
- **imn** (*str*) – Image name
- **dem** (*arr*) – PyTrx.DEM.ExplicitRaster object
- **extent** (*list, optional*) – DEM extent indicator, default to None

Returns A figure with the plotted uv and xyz GCPs

Utilities.**plotInterpolate**(*grid*, *pointextent*, *dem*, *show=True*, *save=None*)
Function to plot the results of the velocity interpolation process for a particular image pair.

Parameters

- **grid** (*arr*) – Numpy grid. It is recommended that this is constructed using the interpolate-Helper function
- **pointextent** (*list*) – Grid extent
- **dem** (PyTrx.DEM.ExplicitRaster) – Underlying DEM for plotting over
- **show** (*bool, optional*) – Flag to denote whether the figure is shown, defatuls to True
- **save** (*str, optional*) – Destination file to save figure to, defaults to None

Returns A figure with interpolated velocity results overlaid onto a given DEM

Utilities.**plotLinePX**(*uv*, *img*, *show=True*, *save=None*)
Plot figure with image overlayed with pixel line features.

Parameters

- **uv** (*arr*) – Input uv coordinates for plotting over image
- **img** (*arr*) – Image array
- **show** (*bool, optional*) – Flag to denote whether the figure is shown, defatuls to True
- **save** (*str, optional*) – Destination file to save figure to, defaults to None

Returns A figure with plotted line measurements overlaid onto a given image

Utilities.**plotLineXYZ**(*xyz*, *dem*, *show=True*, *save=None*)
Plot figure with image overlayed with xyz coordinates representing either areas or line features.

Parameters

- **xyz** (*arr*) – Input xyz coordinates for plotting
- **dem** (PyTrx.DEM.ExplicitRaster) – Underlying DEM for plotting over
- **show** (*bool, optional*) – Flag to denote whether the figure is shown, defatuls to True
- **save** (*str, optional*) – Destination file to save figure to, defaults to None

Returns A figure with plotted lines overlaid onto a given DEM

Utilities.**plotPrincipalPoint**(*camcen*, *img*, *imn*)
Function to show the principal point on the image, along with the GCPs.

Parameters

- **camcen** (*list*) – Principal point coordinates
- **img** (*arr*) – Image array
- **imn** (*str*) – Image name

Returns A figure with the prinicipal point plotted onto the image

Utilities.**plotResiduals**(*img*, *ims*, *gcp1*, *gcp2*, *gcp3*)
Function to plot sets of points to show offsets. This is commonly used for inspecting differences between image GCPs and projected GCPs, e.g. within the optimiseCamera function.

Parameters

- **img** (*arr*) – Image array
- **ims** (*list*) – Image dimension (height, width)
- **gcp1** (*arr*) – Array with uv positions of image gcps
- **gcp2** (*arr*) – Array with initial uv positions of projected gcps
- **gcp3** (*arr*) – Array with optimised uv positions of projected gcps

Returns A figure of an image, plotted with uv gcps, initial projected gcps, and optimised gcps

Utilities.**plotVeloPX**(*uvvel*, *uv0*, *uv1*, *img*, *show=True*, *save=None*)
Plot figure with image overlayed with pixel velocities. UV data are depicted as the uv point in img0 and the corresponding pixel velocity as a proportional arrow (computed using the arrowplot function).

Parameters

- **uvvel** (*arr*) – Input pixel velocities

- **uv0** (*arr*) – Coordinates (u,v) for points in first image
- **uv1** (*arr*) – Coordinates (u,v) for points in second image
- **img** (*arr*) – Image array
- **show** (*bool, optional*) – Flag to denote whether the figure is shown, defatuls to True
- **save** (*str, optional*) – Destination file to save figure to, defaults to None

Returns A figure with plotted point velocities overlaid onto a given image

Utilities.**plotVeloXYZ**(*xyzvel*, *xyz0*, *xyz1*, *dem*, *show=True*, *save=None*)

Plot figure with image overlayed with xyz velocities. XYZ data are depicted as the xyz point in img0 and the corresponding velocity as a proportional arrow (computed using the arrowplot function).

Parameters

- **xyzvel** (*arr*) – Input xyz velocities
- **xyz0** (*arr*) – Coordinates (x,y) for points in first image
- **xyz1** (*arr*) – Coordinates (x,y) for points in second image
- **dem** (PyTrx.DEM.ExplicitRaster) – Underlying DEM for plotting over
- **show** (*bool, optional*) – Flag to denote whether the figure is shown, defatuls to True
- **save** (*str, optional*) – Destination file to save figure to, defaults to None

Returns A figure with plotted points denoting velocities, overlaid onto a given DEM

3.8 Velocity module

The Velocity module handles the functionality for obtaining velocity and homography measurements from oblique time-lapse imagery. Specifically, this module contains functions for: (1) Performing camera registration from static point feature tracking (referred to here as homography); and (2) Calculating surface velocities derived from feature tracking, with associated errors and signal-to-noise ratio calculated. These functions can be performed with either a sparse or dense method, using corner features for tracking in the sparse method and a grid of evenly space points in the dense method.

class Velocity.**Homography**(*imageList*, *camEnv*, *invmaskPath=None*, *calibFlag=True*, *band='L'*, *equal=True*)

Bases: *Images.ImageSequence*

A class for the processing the homography of an image sequence to determine motion in a camera platform. This class treats the images as a contigous sequence of name references by default.

Parameters

- **imageList** (*list*) – List of images, for the ImageSet object
- **camEnv** (PyTrx.CamEnv.CamEnv) – The Camera Environment corresponding to the images, for the PyTrx.Images.ImageSequence object
- **invmaskPath** (*arr, optional*) – The mask for the stationary feature tracking (for camera registration/determining camera homography), default to None
- **calibFlag** (*bool, optional*) – Flag denoting whether images should be corrected for lens distortion, default to True
- **band** (*str, optional*) – String denoting the desired image band, default to 'L' (grayscale)
- **equal** (*bool, optional*) – Flag denoting whether histogram equalisation is applied to images (histogram equalisation is applied if True); default is True.

calcHomographies(*params*, *homogmethod=8*, *ransacReprojThreshold=5.0*)

Function to generate a homography model through a sequence of images, and perform for image registration. Points that are assumed to be static in the image plane are tracked between image pairs, and movement in these points are used to generate sequential homography models. Input example: For sparse homographies: homog = Homography.calcHomographies([['sparse'], [50000, 0.1, 5], [(25,25), 1.0, 4]]) For dense homographies: homog = Homography.calcHomographies([['dense'], [100,100], [cv2.TM_CCORR_NORMED, 50, 100, 1.0, 4]])

Parameters

- **params** (*list*) – List that defines the parameters for point matching: Method: 'sparse' or 'dense' (str). Seed parameters: either containing the cornerparameters for the sparse method - max. number of corners (int), quality (int), and min. distance (int). Or the grid spacing (list) for the dense method. Tracking parameters: either containing the sparse method parameters - window size (tuple), backtracking threshold(int) and minimum tracked features (int). Or the dense method parameters - tracking method (int), template size (int), search window size (int), backtracking threshold (int), and minimum tracked features (int)
- **homogmethod** (*int, optional*) – Method used to calculate homography model, which plugs into the OpenCV function. This can either be cv2.RANSAC (a RANSAC-based robust method), cv2.LEAST_MEDIAN (a Least-Median robust method) or '0' (a regular method using all the points); default to cv2.RANSAC
- **ransacReprojThreshold** (*int*) – Maximum allowed reprojection error, default to 5.0

Returns A list of homography information for all image pairs in sequence

Return type list

getInverseMask()

Return inverse mask.

class Velocity.**Velocity**(*imageList*, *camEnv*, *homography=None*, *maskPath=None*, *calibFlag=True*, *band='L'*, *equal=True*)

Bases: *Images.ImageSequence*

A class for the processing of an ImageSet to determine pixel displacements and real-world velocities from a sparse set of points, with methods to track in the xy image plane and project tracks to real-world (xyz) coordinates. This class treats the images as a contigous sequence of name references by default.

Parameters

- **imageList** (*list*) – List of images, for the PyTrx.Images.ImageSequence object
- **camEnv** (PyTrx.CamEnv.CamEnv) – The Camera Environment object, PyTrx.CamEnv.CamEnv corresponding to the images
- **homography** (*list, optional*) – Homography model for the corresponding image, defaults to None
- **maskPath** (*str, optional*) – The file path for the mask indicating the target area for deriving velocities from. If this file exists, the mask will be loaded. If this file does not exist, then the mask generation process will load, and the result will be saved with this path; default to None
- **calibFlag** (*bool, optional*) – Flag denoting whether images should be corrected for lens distortion, default to True
- **band** (*str, optional*) – String denoting the desired image band, default to 'L' (grayscale)

- **equal** (*bool, optional*) – Flag denoting whether histogram equalisation is applied to images (histogram equalisation is applied if True); default is True.

calcVelocities(*params*)

Function to calculate velocities between succesive image pairs. Image pairs are called from the ImageSequence object. Points are seeded in the first of these pairs using the Shi-Tomasi algorithm with OpenCV's goodFeaturesToTrack function. The Lucas Kanade optical flow algorithm is applied using the OpenCV function calcOpticalFlowPyrLK to find these tracked points in the second image of each image pair. A backward tracking method then tracks back from these to the first image in the pair, checking if this is within a certain distance as a validation measure. Tracked points are corrected for image distortion and camera platform motion (if needed). The points in each image pair are georectified subsequently to obtain xyz points. The georectification functions are called from the Camera Environment object, and are based on those in ImGRAFT (Messerli and Grinsted, 2015). Velocities are finally derived from these using a simple Pythagoras' theorem method. This function returns the xyz velocities and points from each image pair, and their corresponding uv velocities and points in the image plane.

Parameters **params** (*str*) – List that defines the parameters for deriving velocity: Method: 'sparse' or 'dense' (str). Seed parameters: either containing the corner parameters for the sparse method - max. number of corners (int), quality (int), and min. distance (int). Or the grid spacing (list) for the dense method. Tracking parameters: either containing the sparse method parameters - window size (tuple), backtracking threshold (int) and minimum tracked features (int). Or the dense method parameters - tracking method (int), template size (int), search window size (int), correlation threshold (int), and minimum tracked features (int)

Returns A ist containing the xyz and uv velocities. The first element holds the xyz velocity for each point (xyz[0]), the xyz positions for the points in the first image (xyz[1]), and the xyz positions for the points in the second image(xyz[2]). The second element contains the uv velocities for each point (uv[0], the uv positions for the points in the first image (uv[1]), the uv positions for the points in the second image (uv[2]), and the corrected uv points in the second image if they have been calculated using the homography model for image registration (uv[3]). If the corrected points have not beencalculated then an empty list is merely returned

Return type list

getCamEnv()

Returns the camera environment object (CamEnv).

getMask()

Returns the image mask.

Velocity.**apply_persp_homographyPts**(*pts*, *homog*, *inverse=False*)

Funtion to apply a perspective homography to a sequence of 2D values held in X and Y. The perspective homography is represented as a 3 X 3 matrix (homog). The source points are inputted as an array. The homography perspective matrix is modelled in the same manner as done so in OpenCV.

Parameters

- **pts** (*arr/list*) – Input point positions to correct
- **homog** (*arr*) – Perspective homography matrix
- **inverse** (*bool, optional*) – Flag to denote if perspective homography matrix needs inversing, default to False

Returns corrected point positions

Return type arr

Velocity.**calcDenseHomography**(*img1*, *img2*, *mask*, *correct*, *griddistance*, *templatesize*, *searchsize*, *dem*, *projvars*, *trackmethod=3*, *homogmethod=8*, *ransacReprojThreshold=5.0*, *threshold=0.8*, *min_features=4*)

Function to supplement correction for movement in the camera platform given an image pair (i.e. image registration). Returns the homography representing tracked image movement, and the tracked features from each image.

Parameters

- **img1** (*arr*) – Image 1 in the image pair
- **img2** (*arr*) – Image 2 in the image pair
- **mask** (*arr*) – Mask array for image points to be seeded
- **correct** (*list*) – Calibration parameters for correcting image for lens distortion
- **griddistance** (*list*) – Grid spacing, defined by two values representing pixel row and column spacing
- **templatesize** (*int*) – Template window size in im0 for matching
- **searchsize** (*int*) – Search window size in im1 for matching
- **dem** (*arr*) – PyTrx.DEM.ExplicitRaster object
- **projvars** (*list*) – List containing projection parameters (camera location, camera position, radial distortion coefficients, tangential distortion coefficients, focal length, camera centre, and reference image)
- **trackmethod** (*int*) – (str/int): Method for tmeplate matching: cv2.TM_CCOEFF - Cross-coefficient; cv2.TM_CCOEFF_NORMED - Normalised cross-coeff; cv2.TM_CCORR - Cross correlation; cv2.TM_CCORR_NORMED - Normalised cross-corr; cv2.TM_SQDIFF - Square difference; cv2.TM_SQDIFF_NORMED - Normalised square diff
- **homogmethod** (*int, optional*) – Method used to calculate homography model: cv2.RANSAC - RANSAC-based robust method; cv2.LEAST_MEDIAN - Least-Median robust; 0 - a regular method using all the points. Default to None
- **ransacReprojThreshold** (*int, optional*) – default to 5.0
- **theshold** – Threshold for template correlation; default to 0.8
- **min_features** (*int, optional*) – Minimum number of seeded points to track, default to 4

Returns homogMatrix (arr) - The calculated homographic shift for the image pair; src_pts_corr (arr) - original homography points; dst_pts_corr (arr) - tracked homography points; homog_pts (arr) -back-tracked homography points; ptserror (list) - Difference between the original homography points and the back-tracked points; homogerror (list) -Difference between the interpolated homography matrix and the equivalent tracked points

Return type arr/list

Velocity.**calcDenseVelocity**(*im0*, *im1*, *griddistance*, *method*, *templatesize*, *searchsize*, *mask*, *calib=None*, *homog=None*, *campars=None*, *threshold=0.8*, *min_features=4*)

Function to calculate the velocity between a pair of images using a gridded template matching approach. Gridded points are defined by grid distance, which are then used to either generate templates for matching or tracked using the Lucas Kanade optical flow algorithm. Tracked points are corrected for image distortion and camera platform motion (if needed). The points in the image pair are georectified subsequently to obtain xyz points. The georectification functions are called from the Camera Environment object, and are based on those in ImGRAFT (Messerli and Grinsted, 2015). Velocities are finally derived from these using a simple Pythagoras' theorem

method. This function returns the xyz velocities and points, and their corresponding uv velocities and points in the image plane.

Parameters

- **im0** (*arr*) – Image 1 in the image pair
- **im1** (*arr*) – Image 2 in the image pair
- **griddistance** (*list*) – Grid spacing, defined by two values representing pixel row and column spacing.
- **method** (*int*) – (str/int): Method for tmeplate matching: cv2.TM_CCOEFF - Cross-coefficient; cv2.TM_CCOEFF_NORMED - Normalised cross-coeff; cv2.TM_CCORR - Cross correlation; cv2.TM_CCORR_NORMED - Normalised cross-corr; cv2.TM_SQDIFF - Square difference; cv2.TM_SQDIFF_NORMED - Normalised square diff
- **templatesize** (*int*) – Template window size in im0 for matching
- **searchsize** (*int*) – Search window size in im1 for matching
- **mask** (*arr*) – Mask array for masking DEM
- **calib** (*list, optional*) – Calibration parameters, default to None
- **homog** (*list, optional*) – Homography parameters, hmatrix (arr) and hpts (arr), default to None
- **campars** – List containing information for transforming between the image plane and 3D scene: 1. DEM (ExplicitRaster object); 2. Projection parameters (camera location, camera position, radial distortion coefficients, tangential distortion coefficients, focal length, camera centre, and reference image); 3. Inverse projection parameters (coordinate system 3D scene - X, Y, Z, uv0). Default to None
- **theshold** – Threshold for template correlation; default to 0.8
- **min_features** (*int, optional*) – Minimum number of seeded points to track, default to 4

Returns Two lists, 1. containing the xyz velocities for each point (xyz[0]), the xyz positions for the points in the first image (xyz[1]), and the xyz positions for the points in the second image(xyz[2]); amd 2. containing the uv velocities for each point (uv[0], the uv positions for the points in the first image (uv[1]), the uv positions for the points in the second image (uv[2]), and the corrected uv points in the second image if they have been calculated using the homography model for image registration (uv[3]). If the corrected points have not been calculated then an empty list is merely returned

Return type list

Velocity.**calcSparseHomography**(*img1*, *img2*, *mask*, *correct*, *method=8*, *ransacReprojThreshold=5.0*, *winsize=(25, 25)*, *back_thresh=1.0*, *min_features=4*, *seedparams=[50000, 0.1, 5.0]*)

Function to supplement correction for movement in the camera platform given an image pair (i.e. image registration). Returns the homography representing tracked image movement, and the tracked features from each image.

Parameters

- **img1** (*arr*) – Image 1 in the image pair
- **img2** (*arr*) – Image 2 in the image pair
- **mask** (*arr*) – Mask array for image points to be seeded

- **correct** (*list, optional*) – Calibration parameters for correcting image for lens distortion, default to None
- **method** (*int, optional*) – Method used to calculate homography model: cv2.RANSAC - RANSAC-based robust method; cv2.LEAST_MEDIAN - Least-Median robust; 0 - a regular method using all the points. Default to None
- **ransacReprojThreshold** (*int, optional*) – default to 5.0
- **winsize** (*tuple, optional*) – default to (25, 25)
- **back_thesh** – Threshold for back-tracking distance (i.e.the difference between the original seeded point and the back-tracked point in im0); default to 1.0
- **min_features** (*int, optional*) – Minimum number of seeded points to track, default to 4
- **seedparams** (*list, optional*) – Point seeding parameters, which indicate whether points are generated based on corner features or a grid with defined spacing. The three corner features parameters denote maximum number of corners detected, corner quality, and minimum distance between corners; inputted as a list. For grid generation, the only input parameter needed is the grid spacing; inputted as a list containing the horizontal and vertical grid spacing. Default to [50000, 0.1, 5.0]

Returns homogMatrix (arr) - The calculated homographic shift for the image pair; src_pts_corr (arr) - original homography points; dst_pts_corr (arr) - tracked homography points; homog_pts (arr) -back-tracked homography points; ptserror (list) - Difference between the original homography points and the back-tracked points; homogerror (list) -Difference between the interpolated homography matrix and the equivalent tracked points

Return type arr/list

Velocity.**calcSparseVelocity**(*img1*, *img2*, *mask*, *calib=None*, *homog=None*, *invprojvars=None*, *winsize=(25, 25)*, *back_thresh=1.0*, *min_features=4*, *seedparams=[50000, 0.1, 5.0]*)

Function to calculate the velocity between a pair of images. Points are seeded in the first of these either by a defined grid spacing, or using the Shi-Tomasi algorithm with OpenCV's goodFeaturesToTrack function. The Lucas Kanade optical flow algorithm is applied using the OpenCV function calcOpticalFlowPyrLK to find these tracked points in the second image. A backward tracking method then tracks back from these to the original points, checking if this is within a certain distance as a validation measure. Tracked points are corrected for image distortion and camera platform motion (if needed). The points in the image pair are georectified subsequently to obtain xyz points. The georectification functions are called from the PyTrx.CamEnv.CamEnv object, and are based on those in ImGRAFT (Messerli and Grinsted, 2015). Velocities are finally derived from these using a simple Pythagoras' theorem method. This function returns the xyz velocities and points, and their corresponding uv velocities and points in the image plane.

Parameters

- **img1** (*arr*) – Image 1 in the image pair
- **img2** (*arr*) – Image 2 in the image pair
- **mask** (*arr*) –
- **calib** (*list, optional*) – default to None
- **homog** (*list, optional*) – default to None
- **invprojvars** (*list, optional*) – default to None
- **winsize** (*tuple, optional*) – default to (25, 25)
- **back_thesh** – Threshold for back-tracking distance (i.e.the difference between the original seeded point and the back-tracked point in im0); default to 1.0

- **min_features** (*int, optional*) – Minimum number of seeded points to track, default to 4
- **seedparams** (*list, optional*) – Point seeding parameters, which indicate whether points are generated based on corner features or a grid with defined spacing. The three corner features parameters denote maximum number of corners detected, corner quality, and minimum distance between corners; inputted as a list. For grid generation, the only input parameter needed is the grid spacing; inputted as a list containing the horizontal and vertical grid spacing. Default to [50000, 0.1, 5.0]

Returns Two lists, 1. The xyz velocities for each point (xyz[0]), the xyz positions for the points in the first image (xyz[1]), and the xyz positions for the points in the second image(xyz[2]); 2. The uv velocities for each point (uv[0], the uv positions for the points in the first image (uv[1]), the uv positions for the points in the second image (uv[2]), and the corrected uv points in the second image if they have been calculated using the homography model for image registration (uv[3]). If the corrected points have not been calculated then an empty list is merely returned

Return type list

Velocity.**opticalMatch**(*i0*, *iN*, *p0*, *winsize*, *back_thresh*, *min_features*)

Function to match between two masked images using Optical Flow. The Lucas Kanade optical flow algorithm is applied using the OpenCV function calcOpticalFlowPyrLK to find these tracked points in the second image. A backward tracking then tracks back from these to the original points, checking if this is within a given number of pixels as a validation measure. The resulting error is the difference between the original feature point and the backtracked feature point.

Parameters

- **i0** (*arr*) – Image 1 in the image pair
- **iN** (*arr*) – Image 2 in the image pair
- **winsize** (*tuple*) – Window size for tracking e.g. (25,25)
- **back_thesh** – Threshold for back-tracking distance (i.e. the difference between the original seeded point and the back-tracked point in im0)
- **min_features** (*int*) – Minimum number of seeded points to track

Returns Point coordinates for points tracked to image 2 (arr), Point coordinates for points back-tracked from image 2 to image 1 (arr), and SNR measurements for the corresponding tracked point. The signal is the magnitude of the displacement from p0 to p1, and the noise is the magnitude of the displacement from p0r to p0 (arr)

Return type arr

Velocity.**readDEMmask**(*dem*, *img*, *invprojvars*, *demMaskPath=None*)

Read/generate DEM mask for subsequent grid generation. If a valid filename is given then the DEM mask is loaded from file. If the filename does not exist, then the mask is defined. To define the DEM mask, a mask is first defined in the image plane (using point and click, facilitated through Matplotlib Pyplot's ginput function), and then projected to the DEM scene using CamEnv's projectXYZ function. For the projection to work, the invprojvars need to be valid X,Y,Z,uv0 parameters, as generated in CamEnv's setProjection function. The mask is saved to file if a filepath is given. This DEM mask can be used for dense feature-tracking/template matching, where masked regions of the DEM are reassigned to NaN using Numpy's ma.mask function.

Parameters

- **dem** (*arr*) – `PyTrx.DEM.ExplicitRaster` DEM object
- **img** (*arr*) – Image to initially define mask in
- **invprojvars** (*list*) – Inverse projection variables [X,Y,Z,uv0]

- **demMaskPath** (*str, optional*) – File path to outputted mask file, default to None

Returns A Boolean visibility matrix (which is the same dimensions as the dem)

Return type arr

Velocity.**seedCorners**(*im*, *mask*, *maxpoints*, *quality*, *mindist*, *min_features*)

Function to seed corner features using the Shi-Tomasi corner feature detection method in OpenCV's goodFeaturesToTrack function.

Parameters

- **img** (*arr*) – Image for seeding corner points
- **mask** (*arr*) – Mask array for points to be seeded
- **maxpoints** (*int*) – Maximum number of corners detected
- **quality** (*int*) – Corner quality (between 0.0 and 1.0)
- **mindist** (*int*) – Minimum distance between corners
- **min_features** (*int*) – Minimum number of seeded points to track

Returns Point coordinates for corner features seeded in image

Return type arr

Velocity.**seedGrid**(*dem*, *griddistance*, *projvars*, *mask*)

Define pixel grid at a specified grid distance, taking into consideration the image size and image mask.

Parameters

- **dem** (*arr*) – PyTrx.DEM.ExplicitRaster DEM object
- **griddistance** (*list*) – Grid spacing, defined by two values representing pixel row and column spacing.
- **projvars** (*list*) – Projection parameters (camera location, camera position, radial distortion coefficients, tangential distortion coefficients, focal length, camera centre, and reference image)
- **mask** (*arr*) – Mask array for masking DEM

Returns Two arrays containing the grid point positions in the DEM coordinate system (arr), and the grid point positions in the image coordinate system (arr)

Return type arr

Velocity.**templateMatch**(*im0*, *im1*, *uv0*, *templatesize*, *searchsize*, *threshold=0.8*, *min_features=4*, *method=3*)

Function to template match between two images. Templates in the first image (im0) are generated from a given set of points (uv0) and matched to the search window in image 2 (im1). There are a series of methods that can be used for matching, in adherence with those offered with OpenCV's matchTemplate function. After matching, the origin point of each matched template in image 2 is returned, along with the average correlation in each template.

Parameters

- **im0** (*arr*) – Image 1 in the image pair
- **im1** (*arr*) – Image 2 in the image pair
- **uv0** (*tuple*) – Grid points for image 1
- **templatesize** (*int*) – Template window size in im0 for matching
- **searchsize** (*int*) – Search window size in im1 for matching

- **min_features** (*int, optional*) – Minimum number of seeded points to track, default to 4
- **method** (*int*) – (str/int): Method for tmeplate matching: cv2.TM_CCOEFF - Cross-coefficient; cv2.TM_CCOEFF_NORMED - Normalised cross-coeff; cv2.TM_CCORR - Cross correlation; cv2.TM_CCORR_NORMED - Normalised cross-corr; cv2.TM_SQDIFF - Square difference; cv2.TM_SQDIFF_NORMED - Normalised square diff

Returns Point coordinates for points tracked to image 2 (arr), Point coordinates for points back-tracked from image 2 to image 1 (arr), and SNR measurements for the corresponding tracked point where the signal is the magnitude of the displacement from p0 to p1, and the noise is the magnitude of the displacement from p0r to p0 (arr)

Return type arr

CHAPTER

FOUR

PYTRX EXAMPLES

PyTrx comes with working examples, where the toolset has been used to generate glacial datasets. These examples can be adapted and used, and we hope these are especially useful for beginners in coding and glacial photogrammetry.

4.1 Automated area detection

Automated detection of supraglacial lakes, *driver_autoarea.py*

Example driver for deriving changes in surface area of supraglacial lakes captured from Kronebreen, Svalbard, for a small subset of the 2014 melt season. Regions of interest are automatically detected based on differences in pixel intensity and corrected for image distortion. Previously defined areas can also be imported from file (this can be changed by commenting and uncommenting commands in the 'Calculate areas' section). This script uses images from those found in the 'KR5_2014_subset' folder, and camera environment data associated with the text file 'CameraEnvironmentData_KR5_2014.txt.'

4.2 Manual area detection

Manual detection of meltwater plume extents, *driver_manualarea.py*

Example driver for calculating meltwater plume surface extent at Kronebreen, Svalbard, for a small subset of the 2014 melt season. Specifically this script performs manual detection of meltwater plumes through sequential images of the glacier to derive surface areas which have been corrected for image distortion. Images are imported from those found in the 'KR1_2014_subset' folder, and the camera environment associated with the text file 'CameraEnvironmentData_KR1_2014.txt'

4.3 Manual line detection

Manual detection of glacier terminus profiles, *driver_manualline.py*

Example driver for calculating terminus profiles (as line features) at Tunabreen, Svalbard, for a small subset of the 2015 melt season using modules in PyTrx. This script performs manual detection of terminus position through sequential images of the glacier to derive line profiles which have been corrected for image distortion. Images are imported from those found in the 'TU2_2015_subset' folder, and the camera environment associated with the text file 'CameraEnvironmentData_TU2_2014.txt'

4.4 Point georectification

Georectification of calving event point locations, *driver_ptsgeorectify.py*

Example driver which demonstrates the capabilities of the georectification functions provided in PyTrx (which are based upon those available in ImGRAFT). Pre-defined points are imported which denote calving events at Tunabreen, Svalbard, that have been distinguished in the image plane. These are subsequently projected to xyz locations using the georectification functions in PyTrx. The xyz locations are plotted onto the DEM, with the colour of each point denoting the style of calving in that particular instance. The xyz locations are finally exported as a text file (.txt) and as a shape file (.shp).

4.5 Sparse feature-tracking

Glacier velocities derived through feature-tracking of sparse points, *driver_velocity1.py*

Example driver for deriving sparse velocities from Kronebreen, Svalbard, for a small subset of the 2014 melt season. Specifically this script performs feature-tracking through sequential daily images of the glacier to derive surface velocities (spatial average, individual point displacements and interpolated velocity maps) which have been corrected for image distortion and motion in the camera platform (i.e. image registration). This script uses images from those found in the 'KR2_2014_subset' folder, and camera environment data associated with the text file 'CameraEnvironmentData_KR2_2014.txt'.

4.6 Dense feature-tracking

Glacier velocities derived through feature-tracking of dense grid, *driver_velocity2.py*

Example driver for deriving dense velocities from Kronebreen, Svalbard, for a small subset of the 2014 melt season. Specifically this script performs feature-tracking through sequential daily images of the glacier to derive surface velocities (spatial average, individual point displacements and interpolated velocity maps) which have been corrected for image distortion and motion in the camera platform (i.e. image registration). This script uses images from those found in the 'KR2_2014_subset' folder, and camera environment data associated with the text file 'CameraEnvironmentData_KR2_2014.txt'.

4.7 Sparse and dense feature-tracking

Alternative script for glacier velocity feature-tracking with both the sparse and dense methods, *driver_velocity3.py*

Extended example driver for deriving velocities from Kronebreen, Svalbard, for a small subset of the 2014 melt season. This script produces the same outputs as *driver_velocity1.py* and *driver_velocity2.py*. The difference is that velocities are processed using the stand-alone functions provided in PyTrx, rather than handled by PyTrx's class objects. This provides the user with a script that is more flexible and adaptable.

CHAPTER

FIVE

LINKS AND ACKNOWLEDGEMENTS

5.1 PyTrx citations

We are happy for others to use and adapt PyTrx for their own processing needs. If used, please cite the following key publication and Digital Object Identifier:

How et al. (2020) PyTrx: a Python-based monoscopic terrestrial photogrammetry toolset for glaciology. Frontiers in Earth Science 8:21, doi:10.3389/feart.2020.00021

PyTrx has been used in the following publications. In addition to the publication above, please cite any that are applicable where possible:

- How et al. (2019) Calving controlled by melt-undercutting: detailed mechanisms revealed through time-lapse observations. *Annals of Glaciology* 60 (78), 20-31, doi:10.1017/aog.2018.28
- How (2018) Dynamical change at tidewater glaciers examined using time-lapse photogrammetry. PhD thesis, University of Edinburgh, UK, https://hdl.handle.net/1842/31103
- How et al. (2017) Rapidly changing subglacial hydrological pathways at a tidewater glacier revealed through simultaneous observations of water pressure, supraglacial lakes, meltwater plumes and surface velocities. *The Cryosphere* 11, 2691-2710, doi:10.5194/tc-11-2691-2017
- Addison (2015) PyTrx: feature tracking software for automated production of glacier velocity. MSc thesis, University of Edinburgh, UK, https://hdl.handle.net/1842/11794

5.2 Permissions

The DEM of the Kongsfjorden area provided as an example dataset for PyTrx originates from the freely available DEM dataset provided by the Norwegian Polar Institute, data product 'S0 Terrengmodell - Delmodell_5m_2009_13822_33 (GeoTIFF)'. This data is licensed under the Creative Commons Attribution 4.0 International (CC BY 4.0) license:

Norwegian Polar Institute (2014). Terrengmodell Svalbard (S0 Terrengmodell) [Data set]. Norwegian Polar Institute. doi:10.21334/npolar.2014.dce53a47

The two DEMs distributed with PyTrx for the Kongsfjorden region are *KR_demsmooth.tif* and *KR_demzero.tif*, which have been modified and manipulated from the original NPI data. In both cases, the scene has been clipped to the area of interest, downgraded to 20 metre resolution, and smoothed using a linear interpolation method. The latter of these DEMs has been manipulated in order to better represent the terminus position of Kronebreen in 2014 (the time at which the images were taken) and project meltwater plumes to a flat, homogeneous surface at sea level.

The DEM of the Tempelfjorden area provided as an example dataset for PyTrx originates from ArcticDEM, Scene ID: WV01_20130714_1020010 (July 14, 2013). There is no license for the ArcticDEM data and it can be used and distributed freely. The DEM was created from DigitalGlobe, Inc., imagery and funded under National Science Foundation awards 1043681, 1559691, and 1542736.

The DEM distributed with PyTrx of the Tempelfjorden region is called *TU_demzero.tif*, which has been modified and manipulated from the original ArcticDEM data. The scene has been clipped to the area of interest, downgraded to 20 metre resolution, and all low-lying elevations (< 150 m) have been transformed to 0 m a.s.l. in order to project point locations and line profiles to a flat, homogeneous surface at sea level.

5.3 Acknowledgements

This work would not have been possible without the CRIOS (Calving Rates and Impact on Sea Level) project, who own the example image sets distributed with PyTrx.

Parts of the georectification functions in the PyTrx toolbox were inspired and translated from ImGRAFT, a photogrammetry toolbox for Matlab (Messerli and Grinsted, 2015). Where possible, ImGRAFT has been credited for in the corresponding PyTrx scripts (primarily some passages in the CamEnv.py script) and cited in relevant PyTrx publications.

5.4 Links

There are other useful software available for terrestrial photogrammetry in glaciology:

- Pointcatcher: Matlab-based GUI toolbox for feature-tracking and georectification
- ImGRAFT: Matlab toolbox for feature-tracking and georectification
- EMT (Environmental Motion Tracking): GUI toolbox for feature-tracking and georectification
- CIAS: IDL gui for feature-tracking
- PRACTISE: Matlab toolbox for georectification

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

Area, 9

c

CamEnv, 12

d

DEM, 18

f

FileHandler, 20

i

Images, 25

l

Line, 28

u

Utilities, 29

v

Velocity, 32

INDEX

A

apply_persp_homographyPts() (*in module Velocity*), 34
Area
    module, 9
Area (*class in Area*), 9
arrowplot() (*in module Utilities*), 29

C

calcAutoArea() (*in module Area*), 10
calcAutoAreas() (*Area.Area method*), 9
calcDenseHomography() (*in module Velocity*), 34
calcDenseVelocity() (*in module Velocity*), 35
calcHomographies() (*Velocity.Homography method*), 32
calcManualArea() (*in module Area*), 11
calcManualAreas() (*Area.Area method*), 9
calcManualLine() (*in module Line*), 28
calcManualLines() (*Line.Line method*), 28
calcSparseHomography() (*in module Velocity*), 36
calcSparseVelocity() (*in module Velocity*), 37
calcVelocities() (*Velocity.Velocity method*), 34
calibrateImages() (*in module CamEnv*), 14
CamCalib (*class in CamEnv*), 12
CamEnv
    module, 12
CamEnv (*class in CamEnv*), 13
CamImage (*class in Images*), 25
changeBand() (*Images.CamImage method*), 25
checkMatrix() (*CamEnv.CamCalib method*), 12
checkMatrix() (*in module FileHandler*), 20
clearAll() (*Images.CamImage method*), 25
clearImage() (*Images.CamImage method*), 25
clearImageArray() (*Images.CamImage method*), 25
computeResidualsUV() (*in module CamEnv*), 15
computeResidualsXYZ() (*in module CamEnv*), 15
constructDEM() (*in module CamEnv*), 15

D

dataFromFile() (*CamEnv.CamEnv method*), 13
defineColourrange() (*in module Area*), 11
DEM
    module, 18
DEM_FromMat() (*in module DEM*), 18
DEM_FromTiff() (*in module DEM*), 18
densify() (*DEM.ExplicitRaster method*), 18

E

enhanceImage() (*in module Images*), 27
ExplicitRaster (*class in DEM*), 18

F

FileHandler
    module, 20

G

GCPs (*class in CamEnv*), 14
getCalibdata() (*CamEnv.CamCalib method*), 12
getCamEnv() (*Velocity.Velocity method*), 34
getCamMatrix() (*CamEnv.CamCalib method*), 12
getCamMatrixCV2() (*CamEnv.CamCalib method*), 12
getCols() (*DEM.ExplicitRaster method*), 18
getData() (*DEM.ExplicitRaster method*), 18
getDEM() (*CamEnv.CamEnv method*), 13
getDEM() (*CamEnv.GCPs method*), 14
getDistortCoeffsCV2() (*CamEnv.CamCalib method*), 12
getExif() (*Images.CamImage method*), 25
getExtent() (*DEM.ExplicitRaster method*), 18
getGCPs() (*CamEnv.GCPs method*), 14
getImage() (*CamEnv.GCPs method*), 14
getImage() (*Images.CamImage method*), 26
getImageArray() (*Images.CamImage method*), 26
getImageArrNo() (*Images.ImageSequence method*), 27
getImageCorr() (*Images.CamImage method*), 26
getImageEnhance() (*Images.CamImage method*), 26
getImageFileList() (*Images.ImageSequence method*), 27
getImageName() (*Images.CamImage method*), 26
getImageNames() (*Images.ImageSequence method*), 27
getImageObj() (*Images.ImageSequence method*), 27
getImagePath() (*Images.CamImage method*), 26
getImages() (*Images.ImageSequence method*), 27
getImageSize() (*Images.CamImage method*), 26

getImageTime() (*Images.CamImage method*), 26
getImageType() (*Images.CamImage method*), 26
getInverseMask() (*Velocity.Homography method*), 33
getLength() (*Images.ImageSequence method*), 27
getMask() (*Velocity.Velocity method*), 34
getNoData() (*DEM.ExplicitRaster method*), 19
getOGRArea() (*in module Area*), 11
getOGRLine() (*in module Line*), 28
getRefImageSize() (*CamEnv.CamEnv method*), 13
getRotation() (*in module CamEnv*), 16
getRows() (*DEM.ExplicitRaster method*), 19
getShape() (*DEM.ExplicitRaster method*), 19
getZ() (*DEM.ExplicitRaster method*), 19
getZcoord() (*DEM.ExplicitRaster method*), 19

H

Homography (*class in Velocity*), 32

I

imageGood() (*Images.CamImage method*), 26
Images
    module, 25
ImageSequence (*class in Images*), 26
importAreaData() (*in module FileHandler*), 20
importAreaFile() (*in module FileHandler*), 20
importLineData() (*in module FileHandler*), 20
importLineFile() (*in module FileHandler*), 21
interpolateHelper() (*in module Utilities*), 29

L

Line
    module, 28
Line (*class in Line*), 28
lineSearch() (*in module FileHandler*), 21
load_DEM() (*in module DEM*), 19

M

module
    Area, 9
    CamEnv, 12
    DEM, 18
    FileHandler, 20
    Images, 25
    Line, 28
    Utilities, 29
    Velocity, 32

O

opticalMatch() (*in module Velocity*), 38
optimiseCamEnv() (*CamEnv.CamEnv method*), 13
optimiseCamera() (*in module CamEnv*), 16

P

plotAreaPX() (*in module Utilities*), 29
plotAreaXYZ() (*in module Utilities*), 30
plotCalib() (*in module Utilities*), 30
plotGCPs() (*in module Utilities*), 30
plotInterpolate() (*in module Utilities*), 30
plotLinePX() (*in module Utilities*), 30
plotLineXYZ() (*in module Utilities*), 31
plotPrincipalPoint() (*in module Utilities*), 31
plotResiduals() (*in module Utilities*), 31
plotVeloPX() (*in module Utilities*), 31
plotVeloXYZ() (*in module Utilities*), 32
projectUV() (*in module CamEnv*), 16
projectXYZ() (*in module CamEnv*), 17

R

readCalib() (*in module FileHandler*), 21
readDEMmask() (*in module Velocity*), 38
readGCPs() (*in module FileHandler*), 21
readImg() (*in module FileHandler*), 21
readMask() (*in module FileHandler*), 22
readMatrixDistortion() (*in module FileHandler*), 22
reportCalibData() (*CamEnv.CamCalib method*), 12
reportCamData() (*CamEnv.CamEnv method*), 14
reportCamImageData() (*Images.CamImage method*), 26
reportDEM() (*DEM.ExplicitRaster method*), 19
returnData() (*in module FileHandler*), 22

S

seedCorners() (*in module Velocity*), 39
seedGrid() (*in module Velocity*), 39
setColourrange() (*Area.Area method*), 10
setEnhance() (*Area.Area method*), 10
setMax() (*Area.Area method*), 10
setProjection() (*in module CamEnv*), 17
setPXExt() (*Area.Area method*), 10
setThreshold() (*Area.Area method*), 10
showCalib() (*CamEnv.CamEnv method*), 14
showGCPs() (*CamEnv.CamEnv method*), 14
showPrincipalPoint() (*CamEnv.CamEnv method*), 14
showResiduals() (*CamEnv.CamEnv method*), 14
subset() (*DEM.ExplicitRaster method*), 19

T

templateMatch() (*in module Velocity*), 39

U

Utilities
    module, 29

V

Velocity
    module, 32

Velocity (*class in Velocity*), 33
verifyAreas() (*Area.Area method*), 10
voxelviewshed() (*in module DEM*), 20

W

writeAreaCoords() (*in module FileHandler*), 22
writeAreaFile() (*in module FileHandler*), 23
writeAreaSHP() (*in module FileHandler*), 23
writeCalibFile() (*in module FileHandler*), 23
writeHomogFile() (*in module FileHandler*), 23
writeLineCoords() (*in module FileHandler*), 23
writeLineFile() (*in module FileHandler*), 24
writeLineSHP() (*in module FileHandler*), 24
writeVeloFile() (*in module FileHandler*), 24
writeVeloSHP() (*in module FileHandler*), 24