Response to Anonymous Referee #1

The authors describe a toolbox for terrestrial photographs directed towards tidewater glacier outlets. It is a combination of personal best practices of the authors, combining different procedures to extract products from these data. Their targeted audiences seem to be students interested in glaciers, without prior knowledge of computer vision nor photogrammetry. In the wake of open source movements, and the quest for reproducible results the objective of this paper is clear. However, the implementation seems incomplete. If the intended audiences are students, the implementation has serious limitations. Processing of data can be done with PyTrx, but understanding of the limitations might not be gained. If the points given are implemented it might be a worthwhile contribution. However, this is substantial and asks for a complete restructuring of the toolbox.

We would like to thank the reviewer for their constructive comments and feedback. From reading their detailed review, it is clear that the reviewer spent a lot of time reading the paper and testing our toolbox, which we are very grateful for. We believe that this feedback has drastically improved the toolbox, with suggestions that have brought crucial improvements to our attention. Amendments to the toolbox have now been released in our GitHub repository as PyTrx v1.1, with corresponding alterations to the manuscript also. These main changes are:

1. PyTrx has now been re-structured as recommended by the reviewer. PyTrx's core functionality now exists as independent functions which do not depend on inputs from PyTrx's class objects. By doing so, PyTrx is more flexible and easier to adapt to meet users' needs. An example of using PyTrx's independent functions for deriving velocities has been included as an example driver (driver_velocity2.py) to demonstrate this implementation;
2. The Measure.py script has been split according to the functions and class objects for deriving homography, velocity, area and line measurements. We believe this better differentiates the types of measurements that can be derived using PyTrx;
3. Camera calibration functionality has now been added to PyTrx's; functionality. A camera can be calibrated using an inputted set of calibration chessboard images, either using the stand-alone function *calibrateImages* or when initialising the *CamCalib*/*CamEnv* class object;
4. Histogram equalisation is now an optional step, rather than a mandatory step when loading images. This can be toggled using the boolean flag in both the *readImg* function (in FileHandler.py) and the *CamImage* object.

Additionally we have added a section regarding an error budget, as suggested by the reviewer, and endeavoured to find more varied and wide-reaching references.

Details of our response to the reviewer's major and minor comments are outlined subsequently.

Major comments

1. The velocity estimation is based upon optical flow. This technique (especially the Lucas-Kanade implementation) is highly sensitive to intensity changes. When no movement is present, it can still produce velocities due to overcasting. The weakness in this work is that the authors apply histogram equalisation, hereafter optical flow is computed.

An Optical Flow Approximation is adopted in PyTrx's feature-tracking functions because it is highly efficient for tracking a large number of points – 50,000 points can be tracked between a pair of images in under 10 seconds. This is computationally more efficient than using traditional template matching algorithms, and desperately needed in this current age of big data and batch processing. For instance, the template matching function in Python's OpenCV takes 10 times longer to track 2000 points between a pair of images (on average, based on our own testing whilst developing PyTrx).

The drawback of the Optical Flow Approximation approach, as the reviewer rightfully points out, is that matching is based on pixel intensity change rather than relative change. Matching based on relative pixel change in a given region is what makes template matching a robust and reliable method for feature-tracking. The Optical Flow Approximation is highly sensitive to changes in pixel intensity, such as those caused by changes in lighting and shadowing. This can prove challenging when tracking between time-lapse images in glacial environments, where conditions can change very quickly. However, PyTrx accounts for this and is able to limit false tracking due to these sensitives with effective point filtering based on the magnitude and direction of the displacement. This places a heavy reliance on the selection of the images used for feature-tracking, to make sure that images are consistent in lighting and shadowing. However, image selection is a crucial step in all optical image processing techniques due to changes in illumination and shadowing, and their subsequent impact on relative pixel change and the resulting output (e.g. Messerli and Grinsted; 2015; James et al., 2016; Schwalbe and Maas, 2017).

The reviewer suggests that histogram equalisation of the images may hinder trackability with the Optical Flow Approximation technique. Histogram equalisation is applied to all images in PyTrx to enhance image contrast, by adjusting the image intensities. We do this for two reasons:

1. To make corner features in the image more prominent, so more points can be generated/seeded
2. To make features more distinguishable from one image to the next, improving their trackability

In all, we found that applying histogram equalisation improves the distinguishing of glacial features (such as supraglacial lakes and terminus lines) and the tracking of glacial features. This is why it is included in PyTrx as a mandatory step. However, we appreciate that users should have the flexibility to choose whether to apply histogram equalisation to their image sequence. For this reason, PyTrx's histogram equalisation is now optional and is defined throughout (i.e. in all functions and class objects) as an input variable.

2. If the intended audiences are peers, and the toolbox should be seen as a benchmark to build upon, its structure is limited. In such a case one should expect a modular framework where different methodologies can be interchanged. Now, the processing pipelines of the authors are the only pathway, which might not work for other datasets. For example, the supra glacial lake detection is very simple, while more advanced methods already exist (Koschitzki et al. 2014).

Our choice to distribute PyTrx as an object-oriented toolbox stems from the current range of publicly available glacial photogrammetry toolboxes and their flexibility. These toolboxes have been either distributed with a rigid graphical user interface (e.g. Pointcatcher, CIAS) that do not allow access to the source code; or have been distributed as raw functions (e.g. ImGRAFT) which requires background knowledge and labour in order to adapt them to a user's needs.

PyTrx has been designed with object-oriented design in order to provide a middle ground, with semi-rigid functionality. PyTrx has rigidity in its design, catering for beginners in coding and those with little time for adapting raw code. However, the reviewer's comment highlights that PyTrx's flexibility is not adequately demonstrated thus far; rightfully pointing out that PyTrx's core functions are reliant on the class objects.

For this reason, we have totally re-structured PyTrx and released this on our GitHub repository as PyTrx v1.1. This new version now has more flexibility, with PyTrx's core functionality detached from its class objects. Velocities, areas, and line features can now be derived from an image pair/single image without any use of PyTrx's class objects (including image enhancement, georectification, exporting, importing, and plotting functions). We have included an additional example driver (named 'driver_velocity2.py') to demonstrate this, which only uses PyTrx's stand-alone functions to compute glacier surface velocities from a sequence of images. PyTrx's class objects have been adapted for processing measurements from image sequences - effectively, the stand-alone functions are implemented in these class functions and iterated over an entire inputted image sequence. This new flexibility in PyTrx, with the ability to process photogrammetric measurements with both stand-alone functions and class objects, caters for both beginners and advanced computer programmers thereby making PyTrx accessible to all. All of this information has been conveyed in the manuscript with appropriate alterations and added sections, and also updated in the toolbox documentation.

3. Furthermore, a camera calibration procedure is missing in the toolbox, which makes the toolbox appear incomplete.

The reviewer highlights that camera calibration functionality would be an incredibly useful addition to the PyTrx toolbox because it is an integral part of monoscopic photogrammetry. Camera calibration is used to define the intrinsic camera matrix which mathematically represents the camera, and to correct images for distortions that are introduced by the camera and the lens. Raw camera calibration algorithms are available openly, such as the Matlab Computer Vision toolbox and OpenCV for Python and C++. However, these algorithms have yet to be incorporated directly into a glacial photogrammetry toolbox that is openly available to users in the glaciology community. By including them here, PyTrx would be the first open-source glacial photogrammetry toolbox to include camera calibration functionality, to our knowledge. For these reasons, we have now included camera calibration functionality in PyTrx v1.1, based on the algorithms provided in the Python version of OpenCV. This is offered as both a stand-alone function and built into the camera environment (CamEnv) class object. In addition, examples of PyTrx's camera calibration functionality are now included with three of the driver scripts provided – the example for detecting surface lakes ('driver_autoarea.py') and for deriving glacier surface velocities ('driver_velocity1.py' and 'driver_velocity2.py').

Calibration is undertaken in PyTrx using a chessboard/checkerboard approach, which is widely used in computer vision and photogrammetry. The corners of a given chessboard in a set of images are used as a grid to define the intrinsic camera matrix and distortions. Examples of these chessboard images are provided in the Examples directory of PyTrx within the 'calib' folder, which correspond to the two example drivers that perform camera calibrations.

The user can provide the file directory to a set of chessboard images as the calibration input, along with the known number of chessboard corners (i.e. rows, columns), which PyTrx subsequently uses to calibrate the camera. This can either be defined in the camera environment text file (.txt) (whereby the CamEnv object recognises the input based on its data structure and proceeds with camera calibration automatically – or inputted directly to the calibrateImages function.

The calibrateImages function returns the intrinsic camera matrix (K), lens disortion coefficients (k_1, k_2, p_1, p_2, k_3), and the calibration error estimate, which are used subsequently in PyTrx's image correction and georectification functions. The intrinsic camera matrix consists of the focal length in pixels (f_x, f_y) and the principal point (c_x, c_y) as a 3×3 array which is compatible with PyTrx:

$$K = \begin{bmatrix} f_x & 0 & 0 \\ s & f_y & 0 \\ c_x & c_y & 1 \end{bmatrix} \tag{1}$$

Skew (s) is not calculated as part of the intrinsic matrix and is assumed to be 0, as adopted by OpenCV's calibrateCamera algorithm – this is common in computer vision given that camera skew is often neglible in modern cameras. The lens distortion parameters are made up of the radial distortion coefficients (k_1, k_2, k_3) and tangential distortion coefficients (p_1, p_2). Three coefficients are used to represent radial distortion in PyTrx. Whilst we realise that up to eight coefficients can be calculated to define radial distortion, we found that three are more than sufficient and produce the smallest errors for subsequent image correction.

PyTrx's calibrateImages function encapsulates all of the procedures to calibrate a camera, which are primarily taken from the OpenCV toolbox:

1. Chessboard corners are detected in each image (using OpenCV's findChessboardCorners algorithm), based on the inputted chessboard corner dimensions
2. If all corners of the chessboard are found, the locations of these corners are defined in the image plane to sub-pixel accuracy (using OpenCV's drawChessboardCorners and cornerSubPix algorithms)
3. Image plane coordinates for the detected chessboard corners from all imagery are used to calculate the intrinsic camera matrix and lens distortion coefficients (using OpenCV's calibrateCamera algorithm). Firstly, we calibrate a rough camera matrix and distortion coefficients using the raw inputted coordinates. We then optimise these with a second calibration, whereby the principal point that was calculated initally is fixed. By doing this, we refine the camera matrix and distortion coefficients and reduce the errors. From experimenting with this, we found that the principal point is generally the most accurate and reliable output, and therefore we assume that the principal point is correct for the second calibration
4. The optimised camera matrix, lens distortion coefficients, and the error estimate associated with the calibration are returned from this function, which are fed back into the camera environment class object

Subsequent to this, users can now also export the calibration outputs using the writeCalibFile function, which can be found in the FileHandler script. The camera matrix and lens distortion coefficients are exported as a text file (.txt), which is compatible with the calibration file import functionality.

This information has now been incorporated into the manuscript, specifically in Section 4.7 (Image registration and georectification) when discussing camera matrices and lens distortion coefficients.

4. The paper is similar to (Messerli & Grinsted, 2015), therefore the question arises why the authors do not build upon this effort, and instead a new toolbox is introduced. Furthermore, the presented workflow is based upon methodologies used by the authors for other publications. These methodologies are around for quite some time, and thus the presented work does not advance the field nor does it provide new insights.

The reviewer highlights that the methods available in PyTrx (i.e. Optical Flow Approximation, georectification, automated and manual feature detection) have been around for quite some time in photogrammetry and computer vision. However, these methods have not been effectively implemented in glaciology, nor have they been made publicly available to the glaciology community. A small range of toolboxes for glacial photogrammetry applications are currently available, yet none are available in Python which is a common coding language used by the glaciology community. PyTrx provides a valid contribution to glaciology in extending the breadth and range of glacial photogrammetry, and making it more accessible to a greater number in the glaciology and wider environmental science community.

ImGRAFT (Messerli & Grinsted, 2015) is a Matlab toolbox that functions as a feature-tracking tool for optical imagery using template matching. ImGRAFT can calculate glacier velocities from both terrestrial and satellite imagery, with additional georectification algorithms for translating velocities from the terrestrial images. It is a very accomplished toolbox for deriving glacier velocities. However, we wanted to perform additional measurements, namely line measurements (e.g. terminus profiles) and area measurements (e.g. meltwater plume extent). Whilst this can be achieved using ImGRAFT, the functions for doing so are not explicitly provided and it would require a lot of coding knowledge and time to write these. These measurements are not the focus of the ImGRAFT toolbox. PyTrx has therefore been developed to meet this need, and we already have active users who are greatly benefiting from its availability in Python.

5. The authors implement a sparse point cloud. This will result in a scattered data collection of different locations in space and time. While for modelling a fixed coordinate system would be more sufficient, as in (Ahn & Box, 2010).

Traditionally, gridded points are used to track glacier features and generate velocity fields using regularly spaced measurements (e.g. Ahn and Box, 2010; Heid and Kääb, 2012; Messerli and Grinsted, 2015; Schwalbe and Maas, 2017). This is suitable for measuring overall glacier movement in a robust and reliable fashion. However, there is a distinct disadvantage to using gridded points for feature-tracking. Corner features prove most effective for tracking from image to image given that they have highly distinguishable pixel intensities. With gridded points, point selection is based on spacing rather than distinguishable corners thereby limiting effective trackability.

As previously stated, feature-tracking is performed in PyTrx using an Optical Flow Approximation method because it is highly efficient for tracking a large number of points – 50,000 points can be tracked between a pair of images in under 10 seconds. With this many points, a glacier surface can be adequately covered through an image sequence, and produce accurate velocity maps; as demonstrated in Figure 6. These velocity maps have a high spatial resolution that are seldom produced using gridded points (and often take much longer to compute with alternative toolboxes). The reviewer's comment regarding inconsistencies with sparse point clouds are not unfounded, but PyTrx implements thorough filtering (including back-tracking verification) and with careful image selection and a thorough inspection of the output (which should be carried out regardless), we believe the benefits of sparse point tracking far outweigh the limitations.

6. Also an error budget for the 3D transformation is missing, which in the terrestrial setup this scales with distance, see for example (Schwalbe & Maas 2017).

The reviewer highlights the need for an error budget for the georectification functionality within PyTrx. Reviewer #2 also reiterates this, commenting that measurement uncertainty is needed in order to verify PyTrx's capabilities (final comment, Chapter 5). To limit repetition, we have decided to address these two concerns here and summarise: 1) the sources of error introduced with PyTrx's velocity, area and line measurements; and 2) quantification of each error source.

There are a handful of sources of error that are present when deriving measurements from monoscopic terrestrial photogrammetry. Velocity errors have previously been highlighted by Messerli and Grinsted (2015), James et al. (2016), and Schwalbe & Maas (2017) in the presentation of previous glacial photogrammetry toolboxes. These errors, along with the errors for line and area measurements, are outlined here:

1. **Camera motion error**, dictated by the stability of the camera platform and the accuracy of the homography model generated during the image registration process
2. **Pixel tracking error** (in the case of measuring glacier velocity), determined primarily by the selection of the image pair, the coherency of trackable corner features between them, and the magnitude of the pixel track relative to the motion in the camera platform (i.e. the signal-to-noise ratio)
3. **Detection error** (in the case of the automated area detection method), determined primarily by

the variability in illumination/shadowing between the images

4. **Human error** (in the case of the manual detection approaches)
5. **Georectification error**, which is inherently linked to
    (a) The camera model (including focal length and principal point)
    (b) The corrected image, linked to the accuracy of the lens distortion coefficients
    (c) The accuracy of the camera's location and pose (i.e. yaw, pitch, roll)
    (d) The accuracy of the ground control points (GCPs)
    (e) The accuracy of the DEM
    (f) The distance between the camera and the feature of interest

The errors associated with 1, 2, 3 and 4 occur during the measurement in the image plane, with 2 relating to velocity measurements, 3 relating to automated detection, and 4 relating to manual identification. These errors represent the pixel error. The error sources associated with georectification (5) occur during the transformation of these measurements into three-dimensional space (Schwalbe and Maas, 2017). These errors represent the three-dimensional error. Whilst these components have inherent errors associated with them, errors can also be accentuated by inaccurate inputs (such as camera location) and challenging image sequences (e.g. with varying illumination and shadowing). For the purpose of this error analysis, we will therefore look at constraining the errors from each of the outlined components using three of the examples presented in the manuscript, namely the velocities derived from Kronebreen (associated with PyTrx's 'driver_velocity1.py' script), the meltwater plume footprint areas at Kronebreen (associated with PyTrx's 'driver_manualarea.py' script), and the terminus line profiles distinguished at Tunabreen (associated with PyTrx's 'driver_line.py' script). Two measures of error will be calculated – the pixel error from the measurements derived in the image plane, and the three-dimensional error associated with the georectification process.

Error source	**Average velocity error**	**Average area error**	**Average line error**
Camera motion (px)	0.5111	0.1294	0.9863
Pixel tracking (px)	0.9667	–	–
Feature detection (px)	–	86.6870	18.8170
Total pixel error (px)	1.4778	86.8164	19.8033
Total 3D error (%)	0.638	0.638	0.638

Camera motion error is calculated as part of PyTrx's *calcHomography* function, representing the mean error magnitude between an image pair. This error is defined as the movement of static feature points in a pair of images that cannot be accounted for by the homography model (i.e. a RMS value). The error values in the table denote the average error across an image sequence.

The pixel tracking error is associated with calculating velocities through PyTrx's velocity functionality. The error is determined using the back-tracking verification approach discussed in the manuscript. The threshold for back-tracking is set by default to 1. pixel, but can be altered in PyTrx's featureTrack function.

Human error is introduced when a user defines areas/line measurements using PyTrx's manual definition functions. Whilst this error is difficult to constrain, we estimate it for our manual definition examples (in the table above) by iterating the manual definition routine over 10 simulations to produce an average variation. This error will vary between measurements (as demonstrated by the two examples in the table), and therefore it is advised to perform this sensitivity test when using the manual definition methods in PyTrx.

The components that determine the error associated with the georectification process are challenging to constrain individually (e.g. Messerli and Grinsted, 2015; Schwalbe and Maas, 2017). We have chosen to evaluate the these errors collectively through ground-truthing, in order to closely examine the propagation of error over space (i.e. with distance from the camera platform, also known as the baseline). Specifically we used satellite imagery taken at the time of the terrestrial image acquisition to compare the positioning of defined features, namely looking at terminus positions relative to coinciding satellite images (such as those presented in Figure 8).

Over all ground-truthing, we found that the average error (i.e. the difference in position between the georectified feature and the same corresponding feature in the satellite image) was ±0.638%. This error increases with baseline distance, with conservative estimates of 0.2% up to a baseline of 1500 m, and 0.8% for a baseline of 1500–3000 m. For instance, the defined terminus lines depicted in Figure 8 have an average error of ±2.4 m over a baseline of 1500–2000 m, but this error grows to an average of ±7.7 m beyond a baseline of 2000 m. From 3000 m, the error increases exponentially and is difficult to adequately constrain given that our camera set-ups do not cover further than 3500 m.

Similar to error estimations by Messerli and Grinsted (2015) and Schwalbe and Maas (2017), we would advise to adopt the average georectification error estimate (0.638%) for measurements, and use the baseline-specific error estimates in instances where all measurements are localised to a given baseline distance.

This information has now been added to the text, specifically the section regarding the Evaluation of PyTrx (Section 4).

7. Lastly, there is a strong tendency towards referencing to Szeliski, which is a book of references, and a Python image processing book of Solem. Off coarse the authors describe known methodology, but it might have been a bit more specific.

More specific and varied references have now been added to the manuscript.

Minor comments

Page 1, Line 19: 'More toolboxes are therefor needed', I disagree with this argument. It is more worthwhile to extent on previous efforts; open codes are available for Imgraft as well as, photogrammetric libraries such as Ames SP and MicMac.

We suggest in the manuscript that more glacial photogrammetry toolboxes are needed in order to expand the range of toolboxes on offer in different coding languages and with different applications that are beyond calculating glacier velocities. The reviewer rightfully identifies that, in addition to this, it is also worthwhile to focus on expanding pre-established methods.

The reviewer continues by listing examples of 'open code' toolboxes. Whilst we agree that open code is beneficial to users who wish to access and adapt toolboxes, one of the big limitations in monoscopic photogrammetry is the limited range of toolboxes that are open source – i.e. toolboxes that do not require a pricey license in order to operate. For example, ImGRAFT, the photogrammetry toolbox for feature-tracking through satellite scenes and monoscopice set-ups, is programmed in Matlab which requires a license that not all users will have access to.

The reviewer also lists Ames SP and MicMac as other examples of open code toolboxes. Ames SP refers to NASA Ames Stereo Pipeline, which was developed for Multi-View Geometry processing from satellite imagery (Broxton and Edwards, 2008) and has been further applied to generate Digital Elevation Models (DEMs) from stereo satellite imagery (Shean et al., 2016). MicMac is a toolbox for Structure-from-Motion (SfM) processing and has been used for DEM generation from both aerial and terrestrial imagery (Rupnik et al., 2017). These software are open source and require no licensing, but in both these cases their applications are more focused on Multi-View Geometry processing and

Structure-from-Motion rather than monoscopic photogrammetry (i.e. measurements derived from one camera view) in terrestrial settings. As PyTrx is an open-source monoscopic photogrammetry toolbox, multi-camera photogrammetry software (such as Ames SP and MicMac) are incomparable.

For this reason, we have included a passage in the paragraph to express that there should be a greater focus on expanding pre-existing toolboxes, as well as developing new ones: 'In order further glacial photogrammetry techniques, there needs to be greater focus on expanding the capabilities of exisiting toolboxes, and a marked effort to develop new toolboxes which widens the range of data products that can be obtained from time-lapse imagery.' (Page 1, Line 19)

We also stress throughout the manuscript that PyTrx is a monoscopic photogrammetry toolbox (including a change to the title of this manuscript, as suggested by the reviewer) to better convey that this toolbox is not comparable to multi-camera toolboxes and toolboxes that primarily handle satellite imagery.

Page 2, Line 4: 'measurements from photographs' too vague

Wording now changed to make the passage more specific: 'Photogrammetry is defined broadly as the extraction of quantitative measurements from optical imagery...'

Page 2, Line 5: 'photogrammetry' or do you mean signal processing?

Yes, signal processing is part of what is described here, transitioning from earlier traditional techniques to digital signal processing with the introduction of digital cameras and computers with high processing powers. We have now added this to the text.

Page 2, Line 17: 'efficient photgrammetry software', to what extent is PyTrx efficient, there is no emphasis placed in the text about it (batch, multithread,...)

See response to Minor Comment from Reviewer #2 regarding Paragraph 4.6.

Page 2, Line 29: maybe change title to put also an emphasis on monoscopic.

See response to minor comment from Page 1, Line 19.

Page 10, Line 7: 'Matlab Computer Vision toolbox', why is camera calibration not included into PyTrx?

The Matlab Computer Vision toolbox was initally used to calibrate the cameras because it is a well established toolbox that yields reliable and accurate results. We understand that having camera calibration functionality in PyTrx would be ideal and better encapsulate all photogrammetry processing in one unified toolbox. For this reason, we have developed camera calibration functionality in PyTrx using the Python functions available in OpenCV (see major comments for more details about the camera calibration functions available in PyTrx). This has now been updated in the manuscript.

Page 12, Line 9: Why do the authors not use simple functions, this will increase the versatility of the toolbox.

See section 2 of Major Comments.

Page 14, Line 2: Why is there manual inspection? Typically, a dataset has a training and a testing set. Hence, why does PyTrx have not the ability to make a 'ground truth' and then different methodologies can be tested. This reduces the subjectiveness of manual inspection.

Manual inspection is used in PyTrx to better constrain the automated detection of area features from oblique imagery. Area features encapsulate a wide range of glacier features which have different distinguishing properties, such as supraglacial lakes, surfacing meltwater plumes, and debris features. One of the common approaches to distinguishing these in optical imagery is their pixel intensity, however, we realise that this may not necessarily work all the time given optical images are highly sensitive to changes in illumination and shadowing. For this reason, we offer the manual inspection as an optional verify tool to minimise false detection. The automated area detection method has been used to effectively detect supraglacial lake surface areas (e.g. How et al., 2017), and we offer it with the PyTrx toolbox as a broad and basic detecting tool which users can build upon. The use of training and test datasets are application-specific, and we did not see a benefit to including them in the toolbox as this would detract from the broad tools on offer that are presented in this work. The use of training and test datasets, along with the implementation of Machine Learning algorithms, would be an interesting and exciting development, but it is beyond the scope of what is presented here.

Page 14, Line 11: Why not use the HSV space?

PyTrx handles image data in a single band (e.g. grayscale, red, blue, green) in order to uphold computational efficiency and speed. Whilst use of the HSV bands would perhaps be useful to utilise in the detection methods, the main disadvantage is the computational demand in handling three bands of image data. PyTrx's image handling functionality can be altered by the user for this application, but we do not offer it here as it would detract from one of PyTrx's primary benefits, which is its efficiency in batch processing.

Page 15, Line 6: The advantage of Shi-Tomasi is its computational efficiency: the determinant does not have to be calculated

This is helpful information provided by the reviewer which we have now added to the manuscript: 'The Shi-Tomasi Corner Detection method built upon this with a scoring function that does not depend on the calculation of the determinant. Corners are ranked based on the quality level and the minimum Euclidean distance.'

Page 15, Line 17: Why are sparse point clouds used, and why if (Szeliski) is cited consteantly, his adaptive region based selection isn't used? Also, I think most products are more helpful if consistent data points are used, then scattered features, seen throughout a scene.

See Section 5 of Major Comments for reply to the sparse point cloud comment. Reference to Szeliski has been ommited – see Section 7 of Major Comments.

Page 17, Line 5: This is by no means new, the authors might have missed to include (Scambos et al. 1992) & (Jeong et al. 2017).

Agreed. The suggested references have been added to text.

Page 22, Line 2: 'proves to be robust' loose claim, see testing/training comment above.

Phrase removed.

Page 22, Line 11: This backtracking is a relative error. The authors talk about the alternative approach, as implemented by the other toolboxes. These use Monte-Carlo which is an efficient way to grasp propagations of errors, especially in this non-linear system. Thus the authors know of this technique, but implement an inferior method. Why is this done?

See response to Major Comment 6.

Page 22, Line 25: 'toolboxes to choose from', I don't think it is very efficient as a field to have several implementations. All implementing their own best practice, how do the authors see this as a pro?

See section 4 of Major Comments.

Response to Anonymous Referee #2

We would like to thank the reviewer for providing valuable feedback for improving our manuscript. We have provided a thorough response below to each of the reviewer's queries. The main alterations have been the addition of camera calibration functionality to the PyTrx toolbox, the merging of the Introduction and Background sections to avoid repetition, and the inclusion of error estimation (as also requested by Reviewer #1).

Minor comments

Chapter 1 (Introduction) and chapter 2 (background) are clear, but can be merged in a single chapter to avoid some redundancy.

Agreed. Chapter 1 and Chapter 2 have been merged together and sections have been merged to reduce repetition.

Chapter 3 review different aspects to be considered in the post-processing. Anyway, authors refer to many publications issued from their community and should at least cite 2 or 3 papers issued from the image processing community that has work a lot in the past on image calibration and stereo vision (some tools and algorithms mentioned in that paper are issued from this community).

We have now amended this section with more specific and varied references, including work focused on image processing and computer vision.

What should be discussed in that part is the limitation of existing methods versus the specificity of the applied field studied.

We acknowledge that we are discussing monoscopic photogrammetry methods and toolboxes for analysis of glacial imagery and we have now made this clearer at the beginning of Section 3 by changing its title and opening paragraph.

At the end of this chapter, authors should add a synthesis of what is addressed and solved in their present work.

A synthesis has been added to the manuscript, structured as bulleted points to clearly state how PyTrx is unique. However, this has been added to the beginning of Section 4 rather than at the end of Section 3 (as advised) as it was more fitting to add this as an introductory section to the Features and Applications of PyTrx rather than the section on Common Photogrammetric Methods.

'...Specifically, PyTrx has achieved this with the following key features:

1. A sparse feature-tracking approach, using Optical Flow approximation and back-tracking verification to efficiently compute accurate and reliable velocities;
2. Approaches for deriving areal and line measurements from oblique images, with automated and manual detection methods;
3. Camera calibration functionality built in to calculate internal camera matrices and lens distortion coefficients;
4. Written in Python, a free and open-source coding language, and provided with simple example applications for easy use;
5. Engineered with object-oriented design for efficient handling of large data sets;
6. Core methods designed as stand-alone functions which can be used independent of the class objects, making it flexible for users to adapt accordingly.' (Pages 8–9, Lines 28–29 and 1–7)

In chapter 4 : Please do make scheme of your processing workflow as mentioned in the text.

The workflow presented in Figure 4 has been updated to better convey PyTrx's processing workflow, and a reference to it has been added in Section 4.

The camera calibration lean on the use method available in a Matlab Toolbox, which method was used. Why not use available resources in open access and integrate them in your tool ?

The reviewer has highlighted to us that an open-source pathway to producing camera calibration parameters is needed to fully uphold PyTrx's open source ethos. For this reason, we have now included a camera calibration method in PyTrx which can be used either as a stand-alone function, or during the initialisation of the CamEnv class object. For more details, please see Major Comment #3 from Reviewer 1 (who made a similar comment)

Paragraph 4.1 described the data set used with simplifications. What are the consequences of your interpolation for DEM ?

Interpolation of a DEM mimics a smoother, homogeneous surface. This is common practise in glaciology where the date of the DEM acquisition does not exactly match the date of the image acquisition. By creating a smoother, homogeneous DEM, we eliminate artefacts in the glacier surface that do not reflect the surface at the time of the image acquisition. This information has now been added to this section:

'These DEMs are distributed with PyTrx in a modified form, with each scene clipped to the area of interest, downgraded to 20 m resolution, and smoothed using a linear interpolation method. Interpolation is used here to eliminate artefacts in the glacier surface that do not reflect the surface at the time of image acquisition.' (Page 10, Lines 5–8)

Paragraph 4.3 How the authors integrate the natural illumination in their image processing approach? This part could take also benefit of image segmentation by region approaches (see multi spectral imagery).

PyTrx does not contain specific algorithms to correct for changes in natural illumination, like many monoscopic toolboxes for processing glacial imagery (e.g. Pointcatcher, and ImGRAFT) which instead advise selecting imagery at a similar time of day (i.e. similar illumination conditions). Instead, we try to limit false displacements and feature detection (caused by changes in natural illumination and other such factors) by constraining measurements with robust filtering methods such as the back-tracking verification provided in the feature-tracking methods, and the thresholding and verification methods for automated feature detection.

We explored possible approaches for limiting false measurements with multi spectral image analysis, but found that the handling of larger image data detracted from PyTrx's computational efficiency. In all, PyTrx loses its advantage as a toolbox for quick batch processing by implementing this additional analysis. Whilst we agree that this would be a valuable addition, we believe that this is beyond the scope of the work presented here and would be more fitting in a future version of the PyTrx toolbox.

Paragraph 4.4 requires a good stability of the observation sensor used on site.

The authors agree that stability in the camera platform is advantageous when deriving measurements between a pair of images taken at different times. However, platform stability is not vital for seeding points – points are only seeded in one image and therefore shifting in the camera platform does not influence the ability to seed points. We have therefore not included this remark in Section 4.4 (Point Seeding), but have decided to include it in Section 4.5 (Deriving velocities from feature-tracking) where velocities and image registration is discussed as it seems more fitting:

These seeded points are tracked between image pairs in PyTrx using the Lucas Kanade Optical Flow Approximation method (Lucas and Kanade, 1981), which works best when there is minimal motion in the camera platform.' (Page 15, Lines 15–17)

Paragraph 4.6 Analysis of several sequences coupled with obtained results should confirm the efficiency *of the tool. What is the average computing time ?*

Through timing the runtime of PyTrx's example drivers for deriving velocities (using Python's time package), we found that the software can compute the homography (1000 points successfully tracked on average), velocities (30,000 points successfully tracked on average) and georectification of one image pair in 40 seconds, on average. The example driver 'driver_velocity2.py' computes three image pairs in 300 seconds. This was ran on a Linux computer with 7.5 GB of memory. Whilst it is useful to quantify PyTrx's efficiency, it could be misleading as these times are arbitrary and partly depend on the operating system, a computer's memory, and the number of active core processors. For this reason, we have not added this analysis to the manuscript.

Chapter 5 Could take benefit of ground truth using an outdoor controlled environment coupled with an alternative analysis approach for instance using DinSAR or Stereo vision.

The reviewer proposes to add ground-truthing in order to evaluate the accuracy of PyTrx's velocity measurements. Ground-truthing is a viable approach for comparing outputs in order to assess and refine alike measurements. We have implemented ground-truthing to estimate errors from PyTrx's georectification process, as requested by Reviewer #1. The reviewer here suggests ground-truthing PyTrx's velocity results, specifically using an outdoor controlled environment and an alternative processing approach such as DinSAR or stereo vision.

The engineering of an outdoor controlled environment would involve additional field campaigns and primary data collection, shifting the focus of the research presented in this manuscript. Additionally, our studies used monoscopic camera set-ups (i.e. images from one camera) and the use of stereo vision requires a completely different field set-up. Stereo vision is seldom used for deriving glacier velocities because of the difficulties in effectively implementing these field set-ups (e.g. Eiken and Sund, 2012). Therefore stereo vision is not an established approach to conduct ground-truthing, and it is likely that this extra analysis would introduce more uncertainties rather than resolve them. The addition of an outdoor controlled environment and stereo vision in this paper would therefore require a new field campaign to obtain data that is irrelevant to the aim of this paper.

Feature-tracking through optical and SAR satellite imagery is much more common practise for deriving glacier velocities (e.g. Scambos et al., 1992; Heid and Kääb, 2012; Luckman et al., 2015). These methods track large-scale glacier features over large windows/templates, thereby deriving region-based velocities. This region-based tracking approach has also been implemented in glacial photogrammetry toolboxes for deriving velocities from terrestrial imagery (e.g. Messerli and Grinsted, 2015; Schwalbe and Maas, 2017). PyTrx adopts an alternative approach, identifying and tracking individual glacier features on a point-by-point basis. As a result of this, the velocities derived from PyTrx represent small-scale displacements (e.g. surface deformation) as well as region-based movement and is much more fitting for measuring daily and sub-daily displacements. Whilst PyTrx velocities provide highly-detailed displacements that represent localised change, satellite velocities reflect region-based change on a much larger spatial scale. Results from these two approaches are therefore incomparable because they measure different aspects of the glacier system, and ground-truthing with DinSAR velocities would be an unsuitable addition to this research.

Finally, no measurement uncertainty after using this image processing toolbox are mentioned, neither spatial resolution of initial images. Though the authors want to focus on the presentation of their toolbox, it has to be addressed or at least referenced somewhere.

See response to Major Comment 6 from Reviewer #1.

PyTrx: A Python toolbox for deriving glacier velocities, surface areas and line measurements from ~~oblique~~ monoscopic imagery~~in glacial environments~~

Penelope How[1,2*], Nicholas R. J. Hulton[1,2], and Lynne Buie[1]

[1]Institute of Geography, School of GeoSciences, University of Edinburgh, Edinburgh, UK
[2]Department of Arctic Geology, University Centre in Svalbard, Longyearbyen, Norway

Correspondence: Penelope How (p.how@york.ac.uk)

Abstract. Terrestrial time-lapse photogrammetry is a rapidly growing method for deriving measurements from glacial environments because it provides high spatio-temporal resolution records of change. However, glacial photogrammetry toolboxes are limited currently. Without prior knowledge in photogrammetry and computer coding, they are used primarily to calculate ice flow velocities or to serve as qualitative records. PyTrx (available at https://github.com/PennyHow/PyTrx) is presented here as a Python-alternative toolbox to widen the range of monoscopic photogrammetry toolboxes on offer to the glaciology community. The toolbox holds core photogrammetric functions for point seeding, feature-tracking, image registration, and georectification (using a planar projective transformation model). In addition, PyTrx facilitates areal and line measurements, which can be detected from imagery using either an automated or manual approach. Examples of PyTrx's applications are demonstrated using time-lapse imagery from Kronebreen and Tunabreen, two tidewater glaciers in Svalbard. Products from these applications include ice flow velocities, surface areas of supraglacial lakes and meltwater plumes, and glacier terminus profiles.

1 Introduction

Terrestrial ~~time-lapse photogrammetry has proved to be a viable approach for obtaining high spatio-temporal resolution observational records from tidewater glaciers (e.g., Ahn and Box, 2010; Rosenau et al., 2013; James et al., 2014; Pętlicki et al., 2015)~~ photogrammetry is a rapidly growing technique in glaciology as a result of its expanding capabilities, with applications in monitoring change in glacier terminus position (e.g., Kick, 1966), glacier surface conditions (e.g., Parajka et al., 2012; Huss et al., 2013) , supraglacial lakes (e.g., Danielson and Sharp, 2013), meltwater plume activity (e.g., Schild et al., 2016; How et al., 2017; Slater et al., 201 , and calving dynamics (e.g., Kaufmann and Ladstädter, 2008; Ahn and Box, 2010; James et al., 2014; Whitehead et al., 2014; Pętlicki et a . It provides adequate spatial resolution, and the temporal frequency of data-capture is flexible and relatively easy to control. ~~However,~~ A prevailing application has been in deriving glacier surface velocity from sequential monoscopic imagery using a technique called feature-tracking; as it offers highly detailed (both spatially and temporally) records (e.g., Finsterwalder, 1954; Fox et al., 19

*Current address: Department of Environment and Geography, University of York, York, UK

, and a handful of software has been developed to perform feature-tracking through terrestrial, monoscopic time-lapse imagery (e.g., Kääb and Vollmer, 2000; Messerli and Grinsted, 2015; James et al., 2016; Schwalbe and Maas., 2017).

Monoscopic time-lapse photogrammetry remains an under-used technique in glaciology because there are few publicly-available toolboxes for deriving real-world, meaningful measurements from terrestrial imagery. There is an increasing demand for efficient photogrammetry software, which can execute large-batch processing quickly. The majority of ~~these toolboxes have been programmed in one computing language and,~~ monoscopic photogrammetry software are either distributed as raw code or with graphical user interfaces, and without prior knowledge in photogrammetry and computer coding, their applications are largely limited to calculating glacier surface velocities ~~(e.g., Kääb and Vollmer, 2000; Messerli and Grinsted, 2015; James et al., 2016)~~ ~~. More toolboxes are therefore needed to further this technique and its glaciological applications, and add to the~~ (e.g., Kääb and Vollmer, 200 . The future of glacial photogrammetry lies in its valuable ability to examine different aspects of the glacier system simultaneously, such as glacier velocity, fjord dynamics, surface lake drainage and calving dynamics. These can be studied using different image capture frequencies and over different lengths of time. To achieve this, there needs to be greater focus on expanding the capabilities of exisiting toolboxes, and a marked effort to develop new toolboxes which widens the range of data products that can be obtained from time-lapse imagery.

PyTrx (short for 'Python Tracking') is a new toolbox, which is presented here to widen the range of ~~photogrammetric~~ monoscopic photogrammetry toolboxes on offer to the glaciology community, and ~~also~~ expand the types of measurements that can be derived from time-lapse imagery. The toolbox is coded in Python, an open-source computing language, which is freely available and easily accessible to beginners in programming (available at https://github.com/PennyHow/PyTrx). PyTrx has been developed with glaciological applications in mind, with functions for deriving surface velocities via a sparse feature-tracking approach, surface areas (e.g. supraglacial lakes and meltwater plume expressions) with automated area detection, and line profiles (e.g. glacier terminus position) with a manual point selection method.

2 ~~Background~~

~~Photogrammetry is defined broadly as the extraction of measurements from photographs, which may be captured either from above (i.e. aerial) or from the ground (i.e. terrestrial). In recent years, photogrammetry has moved away from traditional techniques (e.g., Finsterwalder, 1954; Kick, 1966), with the introduction of digital cameras and computers with high processing powers.~~

~~Terrestrial photogrammetry is a rapidly growing technique in glaciology as a result of its expanding capabilities, with applications in monitoring glacier surface conditions (Parajka et al., 2012; Huss et al., 2013), supraglacial lakes (Danielson and Sharp, 201~~ ~~, meltwater plume activity (Schild et al., 2016; How et al., 2017; Slater et al., 2017), and calving dynamics (Kaufmann and Ladstädter, 200~~ ~~. A prevailing application has been in deriving glacier surface velocity from sequential terrestrial imagery using a technique called feature-tracking; as it offers highly detailed (both spatially and temporally) records (e.g., Fox et al., 1997; Maas et al., 2006; Dietrich~~ ~~. A handful of software has been developed to perform feature-tracking through terrestrial time-lapse imagery (e.g., Kääb and Vollmer, 2000~~ ~~.~~

~~A key problem is the increasing demand for efficient photogrammetry software, which can execute large-batch processing quickly. The future of its application in glaciology lies in its valuable ability to examine different aspects of the glacier system simultaneously, such as glacier velocity, fjord dynamics, surface lake drainage and calving dynamics. These can be studied using different image capture frequencies and over different lengths of time. To achieve this, the glaciology community needs a greater range of robust photogrammetry methods which have been developed specifically for applications in glaciology.~~

~~Here, a new time-lapse photogrammetry toolbox is presented with specific applications in glaciology. PyTrx has been developed to further terrestrial time-lapse photogrammetry techniques in glaciology and address the issues outlined previously. The software is coded in Python, an open-source computing language, which is freely available and easily accessible to beginners in programming. PyTrx is capable of producing velocities, surface areas and distances from monoscopic time-lapse set-ups. The~~ The common photogrammetry methods used in glaciology will be outlined subsequently, followed by PyTrx's key features and differences. PyTrx's capabilities will be demonstrated and evaluated using time-lapse imagery from Kronebreen and Tunabreen, two tidewater glaciers in Svalbard.

2 Common photogrammetric methods in glaciology

Current photogrammetry software for monoscopic approaches with glacial imagery can generally be divided into those that perform feature-tracking algorithms such as IMCORR (Scambos et al., 1992), COSI-Corr (Leprince et al., 2007), and CIAS (Kääb and Vollmer, 2000; Heid and Kääb, 2012); and those that perform image translation functions such as Photogeoref (Corripio, 2004) ~~, PRACTISE (Härer et al., 2016), Agisoft PhotoScan, and PhotoModeler~~and PRACTISE (Härer et al., 2016) . A common limitation is that few pieces of software unite all the photogrammetry processes needed to compute real world measurements from ~~terrestrial~~ monoscopic time-lapse imagery (i.e. distance, area, ~~velocity, and volume~~and velocity). There are a handful of toolboxes that provide functions for all of these processes, such as the Computer Vision System toolbox for Matlab and the OpenCV toolbox for C++ and Python. However, these are merely ~~given as stand-alone~~ distributed as raw algorithms and a significant amount of time and knowledge is needed to produce the desired measurements and information.

ImGRAFT (available at ~~)and~~ imgraft.glaciology.net), Pointcatcher (available at ~~)~~ lancaster.ac.uk/...pointcatcher.htm) and Environmental Motion Tracking (EMT) (available at at tu-dresden.de/geo/emt/) were the first toolboxes that were made publicly available ~~and~~ that contain all the processes needed to obtain velocities from ~~terrestrial~~ monoscopic time-lapse ~~imagery~~ set-ups in glacial environments ~~(Messerli and Grinsted, 2015; James et al., 2016). Both are Matlab-based toolboxes using algorithms from the Computer Vision System toolbox,~~ (Messerli and Grinsted, 2015; James et al., 2016; Schwalbe and Maas., 2017). These toolboxes have been developed specifically for glaciological applications, either distributed as raw code (in the case of ImGRAFT) or software with a graphical user interface (in the case of Pointcatcher and EMT). These software follow a similar workflow and the steps involved will be discussed subsequently.

2.1 Image processing

Images need to display consistent conditions throughout an image series to gain the best photogrammetric measurements. Therefore, images are commonly enhanced in order to achieve this. Images can be enhanced individually, but this is often time-consuming when handling large image sets. Batch processing is more commonly utilised to enhance images in a time-lapse sequence.

The most straightforward types of enhancements are point operators, where the output value of each pixel in an image depends solely on the corresponding input pixel value, and a given parameter in some cases ~~(Szeliski, 2010)~~(Acharya and Ray, 2005). These include brightness and contrast adjustments, colour corrections. These adjustments do not directly improve the quality of an image though, as changes affect pixel hue and saturation as well as apparent intensity.

Histogram equalisation (also called global histogram equalisation) is generally used to achieve suitable pixel ~~values in an automated manner~~contrast for distinguishing features and reliable tracking (Soha and Schwartz, 1978; Akcay and Avsar, 2017). An intensity mapping function is calculated by computing the cumulative distribution function $(c(I))$ with an integrated distribution $(h(I))$ and the known number of pixels in the image (N) (Solem, 2012):

$$c(I) = \frac{1}{N}\sum_{i=0}^{I} h(I) = c(I-1) + \frac{1}{N}h(I) \tag{1}$$

This reduces the range of pixel values in an image, and smooths drastic changes in lighting and colour.

2.2 Displacement analysis

Displacements are measured through a sequence of images using a technique called feature-tracking, by which pixel intensity features are matched from one image to another ~~(Szeliski, 2010; Solem, 2012)~~(Ahn and Howat, 2011). Pixel-intensity features are defined in the image plane either as points or pixel regions (also referred to as templates), producing spot measurements and continuous surface measurements respectively. These two approaches are also referred to as sparse feature-tracking (e.g., James et al., 2016) and dense feature-tracking ~~.~~ (e.g., Ahn and Box, 2010).

Feature-tracking can be conducted by creating and tracking exclusively between each image pair in an image set (e.g., Messerli and Grinsted, 2015), or by tracking continuously through the image set using the same pixel-intensity features (e.g., James et al., 2016). Whilst tracking continuously enables the calculation of cumulative feature displacements, tracking between each image pair reduces the risk of error propagation from false matches.

Corner objects provide good features to track for sparse feature-tracking methods because they provide distinctive pixel-intensity distributions that can be matched in the subsequent image (Harris and Stephens, 1988). This is demonstrated in Fig. 1, where corner features (e.g. a crevasse corner or debris corner) within an image of a crevasse field prove more distinctive than a homogeneous surface (e.g. bare ice) or an edge feature (e.g. a crevasse edge). These features are marked as points on an image, which can be matched and/or tracked in subsequent images. The creation of these points ~~is called~~ can be referred to as point seeding. Point seeding can be conducted on a manual basis, but this is often time-consuming when handling large image sets (e.g., How, 2013). Automated corner detection methods utilise the high contrast in pixel values to identify the

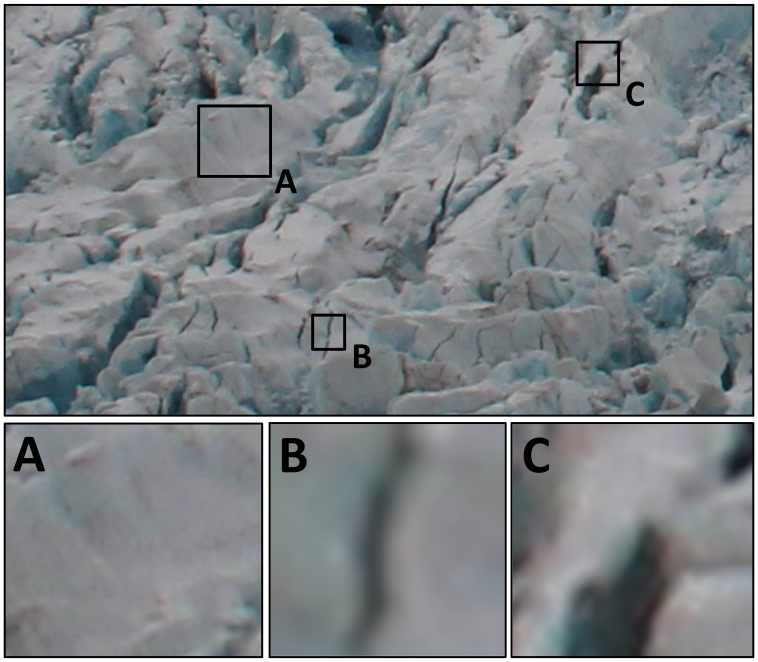

Figure 1. Point seeding coherency demonstrated using regions of a time-lapse image of a crevasse field. The crevasse field (top image) is a subset of a time-lapse image from Kronebreen, Svalbard. The regions highlighted from this subset are examples of a homogenous area (A), an edge feature (B), and a corner feature (C). A has a very indistinctive pixel pattern so would be unable to track from image to image. B has a distinct boundary between different pixel intensities, but tracking may drift along the boundary over time. C is a very distinctive pixel pattern with a defined point that is easy to track.

maximum variation in a given region of the image~~(Szeliski, 2010)~~. There are numerous corner detection methods available for this process, such as the Harris corner detection method (Harris and Stephens, 1988) and the FAST (Features from Accelerated Segment Test) corner detection method (Rosten and Drummond, 2006). The main difference in these methods is precision and efficiency, with trade-offs between ~~the two (Solem, 2012)~~them all.

When tracking between an image pair, the two images have been referred to in many ways, such as 'image A' and 'image B' (e.g., Messerli and Grinsted, 2015), ~~and~~ the 'reference' and 'destination' ~~/' search' images (e.g., Ahn and Box, 2010; James et al., 2016)~~ images (e.g., James et al., 2016), and the 'reference' and 'search' (e.g., Ahn and Box, 2010). The terms reference image and destination image will be used subsequently; the reference image being where points are seeded or templates are defined, and the destination image being where a point/template is matched to. Normalised cross-correlation is a common approach for automated feature-tracking, with cross-correlation referring to the correlation between two signals (i.e. the pixel intensity distribution in two images) (Zhao et al., 2006). This technique is applied in both sparse and dense feature tracking in a similar manner, using the pixel intensity distribution in a window around a given point or a template in the reference image (T)

~~(Szeliski, 2010; Solem, 2012)~~(Solem, 2012):

$$R(x,y) = \frac{\sum_{x',y'} \Big(T(x',y') \cdot I(x+x',y+y')\Big)}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}} \tag{2}$$

Where R is the correlation between the reference template and the destination image, and I is the destination image. The function is applied to each possible position in the image (x,y), thus defining the correlation at every point (x',y'). The highest correlation is defined as the best match between the reference template and the destination image. The correlation at each point in the image can also be determined using different correlation methods such as the normalised square difference, the least square sum and the least difference methods ~~(Szeliski, 2010)~~(e.g., Lowe, 1999).

Tracking coherency through long-duration sequences is frequently subject to severe lighting discrepancies and shadowing which cause false motion. In such applications there is a large significance on image selection. Images with similar lighting and limited shadowing variation must be selected, which may limit the temporal resolution of the collected data. Glacial features which provide good tracking points/templates, such as debris features, can evolve over time which introduce additional displacements (e.g., How, 2013). This can be especially limiting for area-based tracking (e.g., Messerli and Grinsted, 2015). In sparse point tracking, there is a heavy reliance on the number and distribution of points which reduces the replicability of results (Fox et al., 1997; James et al., 2016). It is the factors outlined here that need to be considered when choosing a feature-tracking approach.

2.3 Motion correction

During image acquisition, the time-lapse camera platform is often subject to movement caused by instabilities in the installation, wind, ground heave, thermal expansion of the tripod, and animal/human intervention. This introduces false motion to the measurements derived from an image sequence, which need to be corrected for in order to make accurate measurements through sequential imagery. This process is referred to as image registration.

Feature-based registration methods are more commonly used for glacial environments due to large variations in lighting and glacier surface evolution over time, especially over long-duration sequences ~~(e.g., Messerli and Grinsted, 2015; James et al., 2016)~~ (Eiken and Sund, 2012). Feature-based registration aligns the images by tracking static feature points. These can be natural features, such as mountain peaks (e.g., James et al., 2016), or man-made targets (e.g., Dietrich et al., 2007). Observed movement of these points signify false motion.

Between 20 and 30 coherent natural static feature points can represent false motion effectively. Ideally, these would be distributed evenly across the image plane. Control points in the foreground of an image are more sensitive and can be better for constraining camera rotation angles (Eiken and Sund, 2012), but equally heavy reliance on these can introduce excessive noise and random pixel variation (James et al., 2016).

The two-dimensional pixel displacements between point pairs are subsequently used to calculate motion in the camera, which is represented in three-dimensional space. This translation to three-dimensional space is achieved using a transformation model. There are a number of transformation models in existence, such as isometries, similarity transformations, and affine transformations. Planar projective transformations are used typically for time-lapse photogrammetry in glaciology because they utilise homogeneous coordinates and therefore translate an original planar to a continuous surface. This is also a technique typically used in georectification (see subsequent section for more details).

The point pairs from the static point features are used to map the destination image to the reference image. This map is also referred to as the homography (H), and is computed as follows:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} \quad or \quad x' = Hx \tag{3}$$

Where the h values correspond with the homogeneous transformation of each point within the planar, which is used to translate coordinates from the original image (x, y, w) to the destination image (x', y', w'). In other words, coordinates in the destination (x') are represented by the homography and the corresponding coordinates in the reference image (Hx) (Hartley and Zisserman, 2004).

The homography encapsulates the three-dimensional rotation of the camera platform as movement around its horizontal (yaw), vertical (pitch) and optic axes (roll) ~~(also~~ (Eiken and Sund, 2012). These have also be referred to as omega, phi, and kappa ~~by James et al., 2016)~~(James et al., 2016). Rotation that cannot be accounted for from the two-dimensional displacements is represented as a root-mean-square (RMS) residual pixel value, which is used as a measure of uncertainty. The output rotations can be applied to correct false motion from feature track measurements, which will improve the signal-to-noise ratio.

2.4 3D conversion

Image translation is the process by which measurements in the image plane are translated to real-world measurements. There are several approaches to image translation, the two main ones in glaciology being scale factoring and georectification. A scale factor describes the absolute distance per pixel in an image at a given distance. However, this assumes that the measured displacements are precisely perpendicular to the direction of the camera (e.g., Ahn and Box, 2010). With georectification, the image plane is mapped directly to a real-world coordinate system, and this is more commonly used for measuring displacements that are at an angle to the camera (e.g., James et al., 2016).

The planar projective transformation technique described previously (Equation 3) is also used in image georectification. A homography model is calculated that represents the translation from the image plane to the three-dimensional environment (Hartley and Zisserman, 2004). This is determined using an assortment of information about the three-dimensional environment and how the camera captures this. A Digital Elevation Model (DEM) is typically used to represent the three-dimensional environment. Ground Control Points (GCPs) are point locations in the image plane with corresponding real-world coordinates, which are used as pinning points to map the image to a known coordinate system.

Geometric camera calibration is used to model how the camera captures the three-dimensional environment. There are numerous models which define this translation~~(Solem, 2012)~~. The precedent model used in glaciology utilises the extrinsic

(R,t) and intrinsic (K) information about the camera (Xu and Zhang, 1996):

$$P = \begin{bmatrix} R \\ t \end{bmatrix} K \tag{4}$$

Where P is the camera matrix that mathematically represents the translation between the three-dimensional world scene and the two-dimensional image; R,t are the extrinsic camera parameters that represent the location of the camera in three-dimensional space; and K are the intrinsic camera parameters that represent the conversion from three-dimensional space to a two-dimensional image plane.

The extrinsic and intrinsic camera parameters can be considered as separate matrices~~(Solem, 2012)~~. The extrinsic camera matrix consists of a 3×3 matrix that represents camera rotation (r) and a column vector that represents camera translation (t) ~~:~~ (Zhang, 2000):

$$\left[\begin{array}{c|c} R & t \end{array}\right] = \left[\begin{array}{ccc|c} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \end{array}\right] \tag{5}$$

The intrinsic camera matrix (K) is a 3×3 matrix that contains information about the focal length of the camera in pixels (f_x, f_y), the principal point in the image (c_x, c_y) and the camera skew (s) ~~(Solem, 2012):~~ (Heikkila and Silven, 1997):

$$K = \begin{bmatrix} f_x & 0 & 0 \\ s & f_y & 0 \\ cx & cy & 1 \end{bmatrix} \tag{6}$$

The given focal length for a fixed focal length lens can be assumed to be precise because there is a limited chance of lens drift. It is advised to calculate the focal length for images captured with zoom lenses and compact cameras for greater accuracy. The principal point is also referred to as the optical centre of an image, and describes the intersection of the optical axis and the image plane. Its position is not always the physical centre of the image due to imperfections produced in the camera manufacturing process~~(?)~~; this difference is known as the principal point offset (Hartley and Zisserman, 2004). The skew coefficient is the measure of the angle between the xy pixel axes, and is a non-zero value if the image axes are not perpendicular (i.e. a ‘skewed’ pixel grid).

The intrinsic camera matrix (K) assumes that the system is a pinhole camera model and does not use a lens to gather and focus light to the camera sensor ~~(Solem, 2012)~~(Xu and Zhang, 1996). Camera systems that include a lens introduce distortions to the image plane. These distortions are a deviation from a rectilinear projection, in which straight lines in the real world remain straight in an image. These distortions ineffectively represent the target object in the real world, and therefore distortion coefficients ~~$(k_1, k_2, p_1, p_2, k_3 \ldots k_8)$~~ $(k_1, k_2, p_1, p_2, k_3 \ldots k_5)$ are needed to correct for this. These coefficient values correct for radial ~~$(k_1, k_2, k_3 \ldots k_8)$~~ $(k_1, k_2, k_3 \ldots k_5)$ and tangential (p_1, p_2) distortions ~~(Hartley and Zisserman, 2004):~~ (Zhang, 2000):

$$\begin{aligned} x_{corrected} &= x'\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6 + k_4 r^8 + k_5 r^{10}\right) \\ y_{corrected} &= y'\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6 + k_4 r^8 + k_5 r^{10}\right) \end{aligned} \tag{7}$$

$$x_{corrected} = x' + \left[2p_1xy + p_2\left(r^2 + 2x^2\right)\right]$$
$$y_{corrected} = y' + \left[p_1\left(r^2 + 2y^2\right) + 2p_2xy\right] \tag{8}$$

Where x', y' are the uncorrected pixel locations in an image, and $x_{corrected}, y_{corrected}$ are their corrected counterparts. Radial distortion arises from the symmetry of the camera lens whilst tangential distortion is caused by misalignment of the camera lens and the camera sensor. Radial distortion is the more apparent type of distortion in images, especially in wide angle images, and those containing straight lines (e.g. skyscraper landscapes) which appear curved. Severe tangential distortion can visibly alter the depth perception in images.

The camera matrix and these distortion coefficients can be computed using a set of calibration images taken with the camera that is being used for photogrammetry purposes (Hartley and Zisserman, 2004). These calibration images must be of an object with a known geometry or contain known coordinates (Heikkila and Silven, 1997; Zhang, 2000). A commonly used object is a black and white chessboard, with the positioning and distance between the corners used as the x', y' coordinates~~(Solem, 2012)~~. Other objects which can be used are grids of symmetrical circles (with the centre of each circle forming the x', y' coordinates), and targets which are specified by programs that perform camera calibration (e.g. Agisoft Photomodeller).

The location of the camera and its initial pose (yaw, pitch, roll) are needed to accurately position the camera within the three-dimensional environment. Yaw, pitch and roll are difficult to accurately measure in the field and certain photogrammetry toolboxes ~~calculate and refine yaw, pitch and roll automatically,~~ optimise these parameters, such as ImGRAFT (Messerli and Grinsted, 2015). Camera pose can also be calculated using the principal point (represented as a GCP), along with additional corresponding GCPs to measure the three axes (Fig. 2). The y-axis position of the principal point is used to calculate yaw as an azimuth bearing (Fig. 2A). The xyz position of the principal point provides an elevation comparison to the camera location, which defines the pitch rotation (Fig. 2B). A GCP along the same x-axis position as the principal point is used to calculate roll (Fig. 2C), signified by the apparent change in elevation (Addison, 2015).

3 Features and Applications of PyTrx

PyTrx (available at https://github.com/PennyHow/PyTrx) has been developed to further terrestrial time-lapse photogrammetry techniques in glaciology, and offer an alternative to the monoscopic toolboxes currently available~~, and address the issues outlined previously.~~. Specifically, PyTrx has achieved this with the following key features:

1. A sparse feature-tracking approach, using Optical Flow approximation and back-tracking verification to efficiently compute accurate and reliable velocities;

2. Approaches for deriving areal and line measurements from oblique images, with automated and manual detection methods;

3. Camera calibration functionality built in to calculate internal camera matrices and lens distortion coefficients;

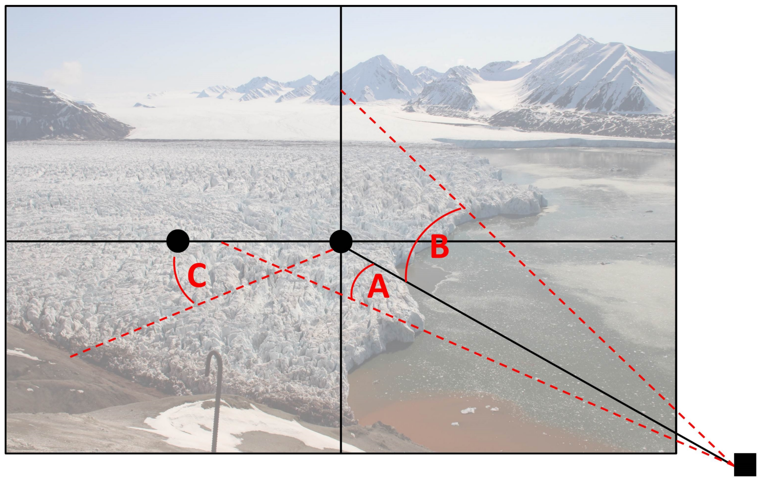

Figure 2. Diagram demonstrating yaw (A), pitch (B), and roll (C) from an image, knowing the camera location (denoted by the square marker) and two GCPs along the x-axis of the principal point (denoted by the two circle markers). The underlying time-lapse image is of the terminus of Kronebreen, Svalbard. Figure adapted from Addison (2015).

4. Written in Python, a free and open-source coding language, and provided with simple example applications for easy use;

5. Engineered with object-oriented design for efficient handling of large data sets;

6. Core methods designed as stand-alone functions which can be used independent of the class objects, making it flexible for users to adapt accordingly.

PyTrx is aimed at ~~beginners in programming, with the files associated with the toolbox written with object-oriented design (i. e. with classes that contain the associated methods and functions). The toolbox follows a similar workflow to that outlined in the previous section.~~ users with all levels of programming experience. Beginners to programming are guided with the rigidity of the class objects, whilst advanced users can use and adapt the stand-alone functions for more complex uses.

3.1 Field set-up

Examples are given throughout this section, which demonstrate the capabilities of PyTrx and its applications in glaciology. These examples use time-lapse imagery collected from Kronebreen (78.8°N, 12.7°E, Fig. 3B) and Tunabreen (78.3^{o}N, 12.3^{o}E, Fig. 3C), which are two tidewater glaciers in Svalbard (Fig. 3A). These time-lapse camera systems consisted of a Canon 600D/700D camera body and a Harbortronics Digisnap 2700 intervalometer, which were powered by a 12 V DC battery and a 10 W solar panel. An assortment of camera lenses was used to allow for flexibility in coverage. These were primarily made up of EF 20 mm f/2.8 USM and EF 50 mm f/1.8 II prime lenses.

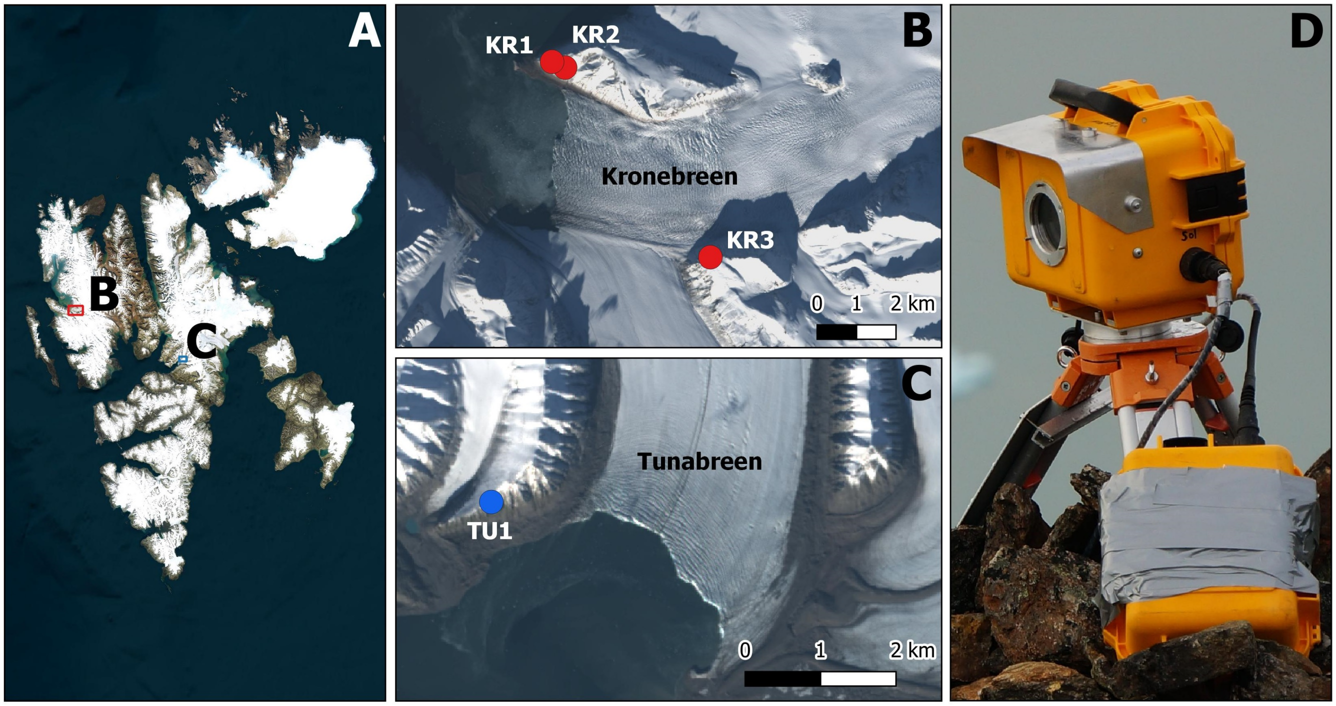

Figure 3. Maps showing the Svalbard archipelago (A); Kronebreen (B) and Tunabreen (C) with numbered camera sites installed over the 2014 and 2015 melt seasons; and an example of one of the time-lapse camera installations in the field (D).

The camera parts and the battery were encased in waterproof Peli Case boxes. The camera enclosures were modified to have a porthole that could hold a sheet of optical glass between two steel frames, through which the camera could take photographs. The camera boxes were fixed on tripods, which were anchored by digging the tripod legs into the ground, burying the tripod legs with stones, and/or drilling guide wires into the surrounding bedrock. An example of one of these set-ups is shown in Fig. 3D.

Accurate locations for each of the time-lapse cameras were measured using a Trimble GeoXR GPS rover to a SPS855 base station, which was positioned ∼15 km away. Positions were differentially post-processed in a kinematic mode using the Trimble Business Centre software, given an average horizontal positional accuracy of 1.15 m and an average vertical positional accuracy of 1.92 m.

GCPs were determined for each camera set-up from known xyz locations that were visible in the field-of-view, and camera pose (yaw, pitch roll) was determined using the approach demonstrated in Fig. 2. Each camera (and lens) was calibrated either using the camera calibration functions in PyTrx (where calibration images are given) or the Matlab Computer Vision ~~Systems~~ toolbox to obtain intrinsic camera matrices and lens distortion coefficient values.

The DEM of the Kongsfjorden area orginates from a freely available DEM dataset provided by the Norwegian Polar Institute, which was obtained from airborne photogrammetric surveying in 2009 (Norwegian Polar Institute, 2014). The DEM of the

Tempelfjorden area originates from ArcticDEM, Scene ID WV01-20130714-1020010 (14th July 2013). These DEMs are distributed with PyTrx in a modified form, with each scene clipped to the area of interest, downgraded to 20 m resolution, and smoothed using a linear interpolation method. Interpolation is used here to eliminate artefacts in the glacier surface that do not reflect the surface at the time of image acquisition. In cases where measurements are derived at sea level (e.g. meltwater plume extents, terminus profiles, and calving event locations), all low-lying elevations (< 150 m) have been transformed to 0 m a.s.l. in order to project them to a flat, homogeneous surface.

3.2 Structure of PyTrx

PyTrx is compatible with Python 2.7 releases, and largely utilises the OpenCV (Open Source Computer Vision) toolbox (v3.1 and upwards), which is a free library designed to provide computer vision and machine learning tools that are computationally efficient and operational in real-time applications. The library has over 2500 optimised algorithms including those for monoscopic photogrammetry and camera calibration (Solem, 2012). A number of other packages are also used, notably GDAL, Glob, Matplotlib, NumPy, ~~OsGeo,~~ and PIL; and these come pre-installed with most Python distributions such as PythonXY and Anaconda.

PyTrx is distributed as a series of files, which requires a driver script to run. The toolbox consists of ~~six~~ eight Python files, which handle the main classes and functions:

1. CamEnv.py – Handles the objects and functions associated with the camera environment;
2. DEM.py – Handles the DEM object and associated functions;
3. Images.py – Handles the objects and functions associated with the image sequence and the individual images within that sequence;
4. ~~Measure~~Velocity.py – Handles the ~~objects associated with making measurementsfrom the images (i.e. velocities, areas, and distances)~~Velocity object and functions for deriving velocity measurements;
5. Area.py – Handles the Area object and functions for deriving areal measurements;
6. Line.py – Handles the Line objects and functions for deriving line;
7. FileHandler.py – Contains ~~all functions called by an object to import and export~~ the functions for importing and exporting data;
8. Utilities.py – Contains ~~all~~ the functions for plotting and presenting data.

Within these files, ~~six~~ nine class objects perform the core photogrammetry processes that were outlined in the previous section: *GCPs*, *DEM*, *CamCalib*, *CamImage*, *ImageSequence*, *Velocity*, *Area*, *Length*, and *CamEnv*. They operate according to the workflow presented in Fig. 4.

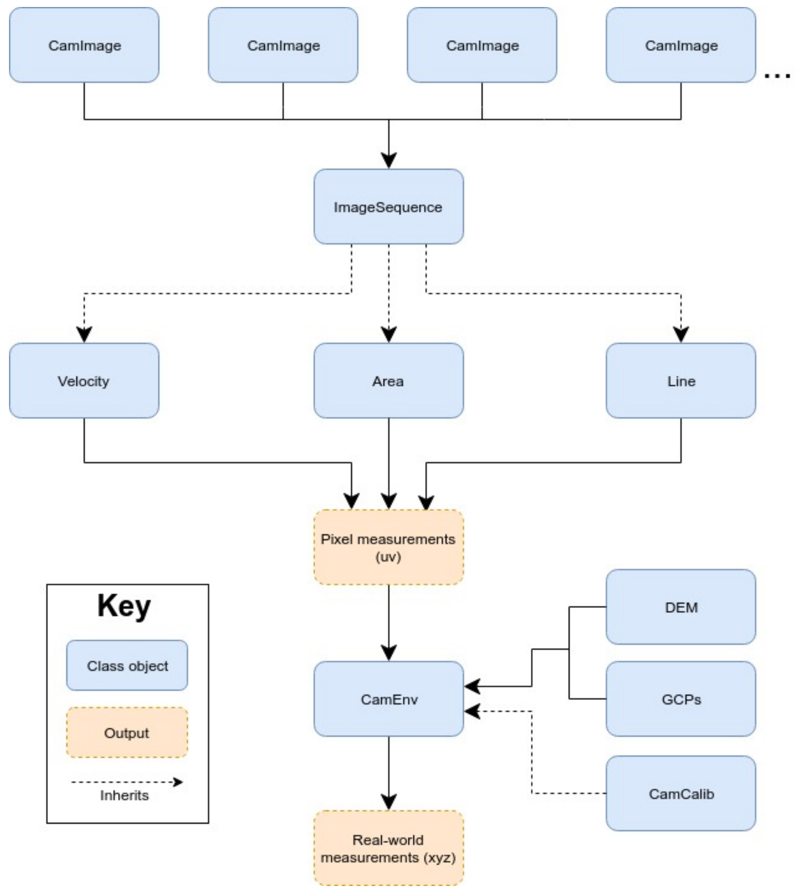

Figure 4. PyTrx's workflow, showing how each of the class objects interact with one another.

The key features of PyTrx, which are different from the general techniques discussed previously, will be outlined comprehensively in the subsequent sections with reference to PyTrx's object-oriented workflow. Class objects, functions and input variables in PyTrx will be highlighted in italics.

3.3 Image enhancement

~~*CamImage* (found in Images.py) holds all the information about a single image within an image sequence, making single images easy to call within PyTrx. Each image is represented as a NumPy arrayin the CamImage class object .~~ An image is passed into PyTrx as an array, either using the *readImg* function found in Filehandler.py or when initialising the *CamImage* object which is in Images.py. Image enhancement processes ~~can be executed within the CamImage object~~ are executed by modifying the ~~NumPy~~ array that represents the image. The image enhancement methods that are available in PyTrx are histogram equalisation (as presented in Section 2.1)~~and~~ , the extraction of information from a single image band or grayscale, and simple arithmetic manipulations on an image's pixel values.

~~One problem with histogram equalisation is that the resulting histogram is flat (Solem, 2012).~~ Grayscale, equalised images are used commonly in photogrammetric processing in order to reduce processing time (e.g., James et al., 2016). This means that all the RGB information is flattened and each pixel is assigned one single grayscale value. However, this can alter the image and its uses for extracting measurements from. For instance, corner coherency can be altered, which can present challenges in point seeding, image registration, and feature tracking ~~(Solem, 2012)~~(Schwalbe and Maas., 2017). Additionally, it can make it difficult to distinguish areas of interest based on pixel intensity.

PyTrx has been designed to overcome this limitation by providing a method for extracting information from a specified band of an image. An image can be passed forward either in grayscale, or with one of the RGB bands. Although this is executed in the *readImg* function or the *CamImage* ~~class~~ object, it can also be defined in the ~~three *Measure* classes (i.e.~~ *Velocity*, *Area*, and ~~*Length*~~*Line* ~~)~~ objects with the *band* variable. The string inputs *r*, *g*, *b*, and *l* denote whether the red, green, blue, or grayscale bands should be passed forward. This does not affect the processing time drastically, and enables effective detection of areas of interest in images, such as meltwater plumes and supraglacial lakes.

~~The user can select an image band to suit their applications, and obtain accurate measurements from an appropriate image band. The~~ The example in Fig. 5 demonstrates how each ~~selection~~ selected band affects the pixel intensity range associated with a cluster of supraglacial lakes. These surface areas have been detected automatically based on pixel intensity.

~~An example of PyTrx's ability to the extract pixel information from a specified image band using an example image from Kronebreen camera K3. The image shows a cluster of supraglacial lakes, which were monitored through the 2015 melt season. A shows the original time-lapse image. The yellow box denotes the subset from which pixel information is extracted from and displayed in the subsequent images: grayscale (B), the red image band (C), the green image band (D), and the blue image band (E). The white plotted lines in these subsets show attempts to automatically detect the lake extent. The red image band yields the best detection as it closely follows the lake extent.~~

Ideally, regions of interest in an image are effectively detected when they are represented by the smallest range in pixel intensity (i.e. a homogeneous surface). ~~With the functionality provided in PyTrx, an assessment can be made to decide which image band facilitates effective detection.~~ In the example presented in Fig. 5, the red image band (Fig. 5C) offers the smallest pixel intensity range and thus the lakes are represented as a homogeneous surface. This proves easiest to define on an automated basis.

~~The *Area* class object~~

PyTrx offers an additional enhancement process to improve the ability to automatically detect areas in images. The ~~image enhancement process within the *Areaclass object is a function that*~~ *enhanceImg* function in Images.py uses simple arithmetic to manipulate image brightness and contrast. This is performed in ~~a NumPy array, and the array~~ an array, which can either represent an image with pixel values from the red, green or blue band, or from the grayscale image; as specified in the *band* variable. This enhancement method uses three variables to change the intensity and range of the pixel values:

1. *diff*: Changes the intensity range of the image pixels. This has two outcomes. Either it changes dark pixels to become much brighter and bright pixels become slightly brighter, or it changes dark pixels to become much darker and bright pixels become slightly darker

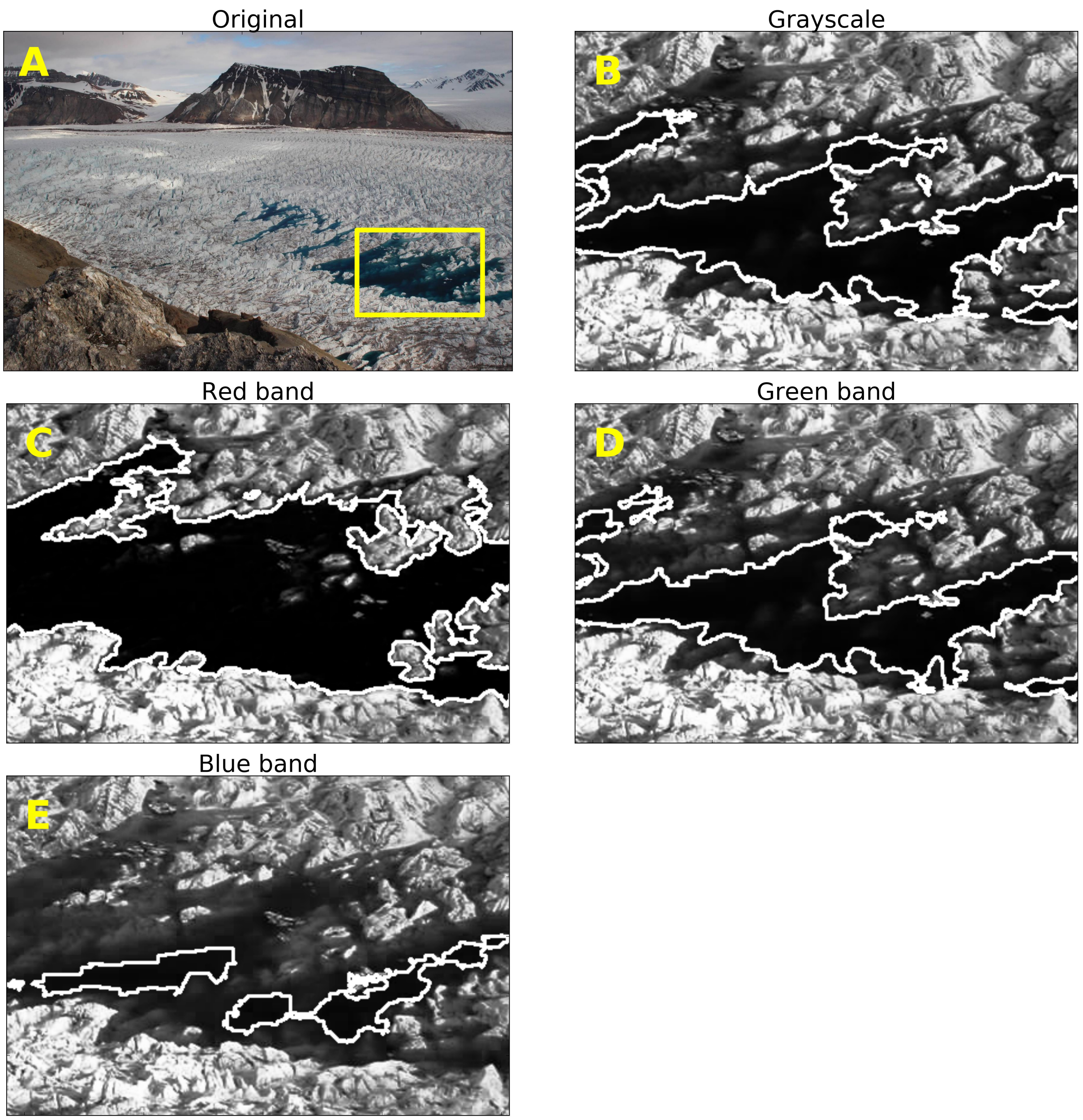

Figure 5. An example of PyTrx's ability to the extract pixel information from a specified image band using an example image from Kronebreen camera K3. The image shows a cluster of supraglacial lakes, which were monitored through the 2015 melt season. A shows the original time-lapse image. The yellow box denotes the subset from which pixel information is extracted from and displayed in the subsequent images: grayscale (B), the red image band (C), the green image band (D), and the blue image band (E). The white plotted lines in these subsets show attempts to automatically detect the lake extent. The red image band yields the best detection as it closely follows the lake extent.

2. *phi*: Modifies the intensity of all pixel values

3. *theta*: Defines the number of 'colours' in the image by grouping pixel intensity regions together i.e. an input of 3 signifies that all the pixels will be grouped into one of three pixel values

The result better distinguishes areas of interest, and makes it easier for the subsequent detection. See Section 3.6 for more information on how areal measurements are derived from images using PyTrx.

3.4 Point seeding

Points are seeded in an image using the Shi-Tomasi Corner Detection method (Shi and Tomasi, 1994), as part of the *goodFeaturesToTrack* function in the OpenCV library. This is based on the Harris Corner Detection method, which evaluates the difference in intensity for a displacement of (u,v) in all directions for a given region of an image (E) (Harris and Stephens, 1988). Points are selected based on the largest intensity differences:

$$E(u,v) = \sum_{x,y} w(x,y)\Big[I\big(x+u,y+v\big) - I\big(x,y\big)\Big]^2 \tag{9}$$

Where w is the window function (rectangular or Gaussian) defined as a width and height (x,y) and I is pixel intensity. The first part of the bracketed section, $(I(x+u,y+v))$ defines the shifted intensity, and the second part $(I(x,y))$ calculates the intensity at the centre origin. A scoring function (R) is subsequently used to define whether the pixel-intensity signature represents a corner, a flat area or an edge:

$$R = \lambda_1\lambda_2 - k\Big(\lambda_1 + \lambda_2\Big)^2 \tag{10}$$

Where k is a tuneable sensitivity parameter, and λ_1 and λ_2 are the eigen values in the x and y axes of a given symmetric matrix ~~(Solem, 2012)~~(Shi and Tomasi, 1994). This forms a descriptor for the matrix, which can be used to evaluate a given region of an image:

1. A ~~corner~~ homogenous ('flat') region of the image is present if ~~λ_1 and λ_2 are both large positive values~~ $\lambda \approx \lambda_2 \approx 0$ (e.g. Fig. 1~~C~~A);

2. An edge is present if one of the eigenvalues is large and the other is approximately zero (e.g. $\lambda_1 > 0$ and $\lambda_2 \approx 0$) (e.g. Fig. 1B);

3. A ~~homogenous ('flat') region of the image~~ corner is present if ~~$\lambda \approx \lambda_2 \approx 0$~~ λ_1 and λ_2 are both large positive values (e.g. Fig. 1~~A)~~C).

Changes between λ_1 and λ_2 can be amplified by modifying k, the sensitivity parameter (Harris and Stephens, 1988). The Shi-Tomasi Corner Detection method built upon this with a ~~further~~ scoring function that ~~ranks the best corner features~~ does not depend on the calculation of the determinant. Corners are ranked based on the quality level and the minimum Euclidean distance (Shi and Tomasi, 1994):

$$R = min\Big(\lambda_1\lambda_2\Big) \tag{11}$$

The quality level denotes the minimum quality of a corner and is measured as a value between 0 and 1, and the function returns the remaining strongest corners. This point seeding method is part of the *featureTrack* function, which is found in Velocity.py. The function attempts to seed 50,000 points with a quality level of 0.1 and a minimum Euclidean distance of 3 pixels) in an image. These default settings produce a heavily populated sparse point set in the image plane.

~~An imagecan be called using~~ Points can be seeded through a series of images by calling each one-by-one through the *ImageSequence* object ~~within PyTrx~~ (found in Images.py). An *ImageSequence* object holds the information about a series of *CamImage* objects (as shown in Fig. 4). It references the *CamImage* objects sequentially (sorted alphabetically, based on the file name) so that they can be called easily in subsequent processing, such as the selection of single images and image pairs. ~~Images are called in this manner to seed points within. PyTrx attempts to seed 50,000 points (with a quality level of 0.1 and a minimum Euclidean distance of 3 pixels) in an image. This can be adjusted accordingly throuh PyTrx's *featureTrack* function, which is found in the *Velocity* object in Measure.py. These default settings produce a heavily populated sparse point set in the image plane.~~

3.5 Deriving velocities from feature-tracking

~~*Velocity* (found in Measure.py)~~ Velocity.py contains all the processing steps for ~~*point seeding*, sparse *feature-tracking*, and *image registration*~~ point seeding, sparse feature-tracking, and image registration between image pairs. The stand-alone functions calculate this information for an image pair, and the *Velocity* object can be used to iterate these functions across a sequence of images.

Points are seeded using the Shi-Tomasi Corner Detection method discussed previously. These seeded points are tracked between image pairs in PyTrx's *featureTrack* function using the Lucas Kanade Optical Flow Approximation method (Lucas and Kanade, 1981), which works best when there is minimal motion in the camera platform. These are verified using a back-tracking technique (used in the *calcVelocity* function, and the *Velocity* object's *calcVelocities* function). The retained points are reprojected subsequently using the ~~*georectification* functions held within the~~ georectification functions in CamEnv.py, which can either be conducted using inputted information about the DEM and camera or called from the *CamEnv* ~~class~~ object. This approach is adopted also for image registration (the ~~*image registration*,~~ *calcHomography* function, and the *Velocity* object's *calcHomographyPairs* function), which uses control points seeded and tracked between image pairs prior to deriving velocities. An example of this feature-tracking and georectification functionality is shown in Fig. 6, demonstrated by deriving surface velocities from an image pair captured at Kronebreen.

Optical Flow is the pattern of apparent motion of an object between two images, caused by the movement of the object or the camera. It is a concept readily employed in video processing to distinguish motion and has been used in motion detection applications to predict the trajectory and velocity of objects (e.g., Baker et al., 2011; Vogel et al., 2012).

Optical Flow is represented as a two-dimensional vector field working on the assumptions that the pixel-intensity distribution of an object does not change between the image pair and the neighbouring pixels display similar motion (Tomasi and Kanade, 1991). Between two images, the position of a pixel (I) will change $(\delta x, \delta y)$ over time (δt), assuming that the pixel intensity is

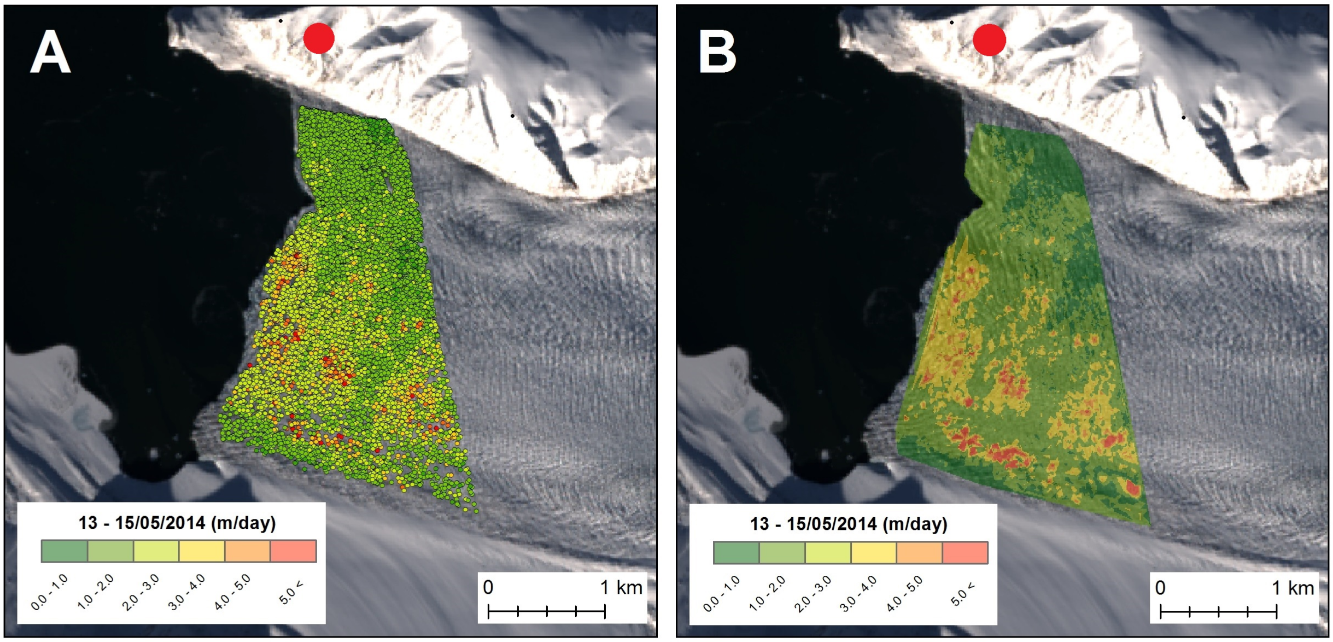

Figure 6. An example of PyTrx's feature-tracking and georectification functionality. Points have been tracked between an oblique time-lapse image pair taken between 13 and 15 May 2014 at Kronebreen camera K2. The raw point positions and associated velocities (A) can be interpolated to create velocity maps of a given area (B). The red point denotes the location of the time-lapse camera (Kronebreen camera K2). This example is provided with PyTrx at https://github.com/PennyHow/PyTrx.

unchanging (Zhang and Chanson, 2018):

$$I(x,y,t) = I\big(x+\delta x, y+\delta y, t+\delta t\big) \tag{12}$$

Although it is mainly used to predict the velocity and trajectory of an object in an image (e.g., Horn and Schunck, 1981), it can also be used for tracking motion between images, such as feature-tracking on glacier surfaces~~(Solem, 2012)~~.

The Lucas-Kanade algorithm approximates the Optical Flow of sparse points from image to image using a search window that is typically 3×3 pixels (Lucas and Kanade, 1981), assuming that all 9 points in each window have the same motion. It is effective at measuring small displacements because of its thorough examination in a given window. This is ideal when dealing with slow-moving glaciers or high interval frequency sequences (e.g. more than one image every day). However, ice velocities derived using this method (such as those shown in Fig. 6) also include signals of crevasse propagation and surface deformation. This is a key reason why dense feature-tracking methods are often preferred for deriving ice velocities over sparse methods; as the template grid ensures that measured displacements are purely associated with ice motion.

The Lucas Kanade Optical Flow approximation algorithm is available in the *calcOpticalFlowPyrLK* function in the OpenCV library, which is used in PyTrx because of its computational efficiency. ~~It is often used in real-time video photogrammetry.~~

~~Tracking is implemented between each image pair, rather than continual tracking through the sequence. The first image in each pair is assigned as the reference image where points are seeded, and the second image is the destination image.~~

As noted in the previous section, photogrammetric measurements can be non-replicable because of the inclusion of falsely tracked points due to a lack of robust point tracking evaluation. Back-tracking verification (Kalal et al., 2010) is implemented in PyTrx to limit false tracking. Back-tracking verification is used to assess point coherency by tracking points back from the destination image to the reference image (e.g., Scambos et al., 1992; Jeong et al., 2017). This generates two sets of points in the reference image, the initial seeded points and the corresponding back-tracked points. If a back-tracked point is within a given distance to the position of the seeded point then it is deemed accurate and is kept. Points which exceed this distance threshold are discarded. The distance between the seeded point and the back-tracked point is used as a measure of noise. Together with the signal, this can be used to determine the signal-to-noise (SNR) ratio of each point.

3.6 Deriving area and line objects

Current photogrammetric software focus on deriving velocities from time-lapse sequences, such as ImGRAFT (Messerli and Grinsted, 2015)~~and Pointcatcher (James et al., 2016)~~, Pointcatcher (James et al., 2016) and EMT (Schwalbe and Maas., 2017). Other measures of the glacial system would be valuable, such as changes in supraglacial lakes, the expression of a meltwater plume, and terminus position. PyTrx has been developed to offer these additional photogrammetric measurements. It can specifically derive area and line/distance measurements, in addition to velocities, from ~~time-lapse~~ image sequences. The *Area* and *Length* class objects contain all of the processing steps to obtain these measurements~~.~~, which can be found in the Areas.py and Line.py scripts, respectively. Both class objects inherit from the ~~*Velocity*~~*ImageSequence* class object.

~~*Area* (found in Measure.py)~~ The Area.py script contains all the processing steps for deriving area measurements from imagery, in both an automated and manual manner. The stand-alone functions provide methods for measuring areas from a single image, whilst the *Area* object can be used to measure areas across a series of images. The automated detection of areas is based on changes in pixel intensity, from which points are seeded around a detected extent. ~~*Length* (found in Measure.py)~~ The Line.py script contains all the processing steps for ~~manual line /distance~~ manually deriving line measurements from imagery~~. This is done using the same technique found in the *Area* class object, from which the textitLengthclass object inherits from. The points~~, with stand-alone functions for making measurements from a single image and the *Line* object for making measurements across a series of images. The area and line features defined in both these ~~methods can be~~ scripts are translated to real-world ~~area and line objects using the *georectification* information and functions in~~ measurements using the georectification functions in CamEnv.py, in a similar fashion to the ~~*CamEnv* class object.~~approach in Velocity.py.

~~Functions in the *fileHandler* file can be used to export the data from the area and line objects.The information can be exported as listed point coordinates and areas/distances using the *writeAreaFile* and *writeLineFile* functions. Additionally, the objects can be exported as .shp files using the *writeSHPFile* function for easy importing into mapping software such as ArcMap and QGIS.~~

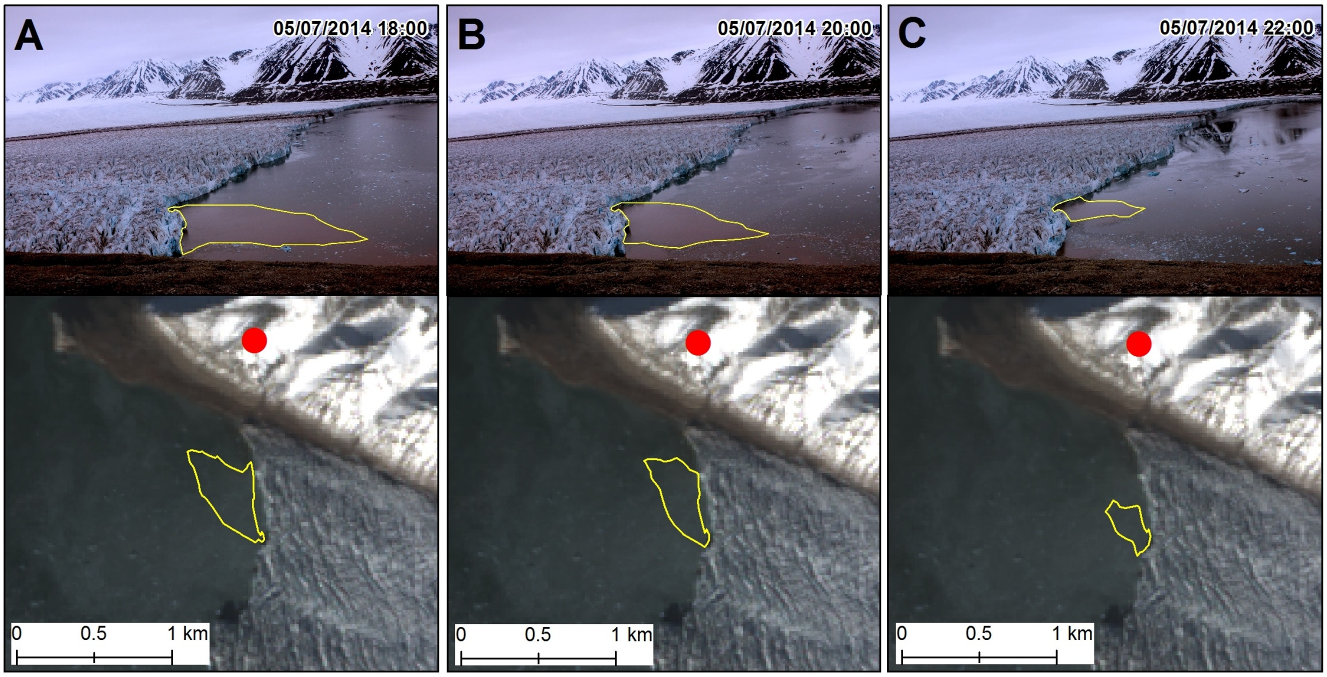

Figure 7. Changes in meltwater plume extent distinguished from time-lapse imagery of Kronebreen camera K1. The surface expression of the meltwater plume has been tracked through images captured on 05 July 2014 at 18:00 (A), 20:00 (B), and 22:00 (C) to demonstrate its diurnal recession. Each plot shows the plume definition in the image plane (top) and its translation to real-world coordinates (bottom). A similar example of this is provided with PyTrx at https://github.com/PennyHow/PyTrx.

~~Shapefiles~~ Areal features have been constructed from the distinguished surface expression of meltwater plumes and overlaid onto a ~~Landsat~~ satellite scene in the example presented in Fig. 7. Meltwater plumes are the main sources of outflow from a tidewater glacier, and tracking their surface area can be used to infer changes in discharge (e.g., How et al., 2017). The steady recession of the meltwater plume extents shown in Fig.7 is linked to diurnal fluctuations in melt production.

3.6.1 Automated detection

Areas are automatically computed using the *calcAutoArea* function and *Area* ~~contains functions for automated detection of areas. Areas~~ object's *calcAutoAreas* function in Area.py. These are automatically detected based on pixel intensity within the image plane. This entails several key steps and functions to ensure adequate detection in each image. The image is masked to the area of interest firstly, thus reducing processing time and limiting the chance of false detection. The mask can be defined or read from file using the *readMask* function, which is within the FileHandler.py ~~file. This mask~~ script. This masking function is also used for seeding points in a given region of an image (as part of the feature-tracking and image registration functionality).

The image is next subjected to the simple arithmetic enhancement method outlined in Section 3.3 to better distinguish the target area. The pixel intensities associated with the target area are defined subsequently as a range of the lowest and highest values. This range can either be pre-defined, or manually defined within the program on a point-and-click basis (using the *defineColourrange* function). Pixels within this intensity range are distinguished and grouped using the OpenCV function *cv2.inRange*. The grouped pixels form regions which are transformed into polygons using the OpenCV function *cv2.contour*. Each point within the polygon(s) is defined by coordinates within the image plane.

Often this results in many polygons being created. The number of points in each polygon is used to filter out noise and falsely-detected areas, with small polygons (i.e. constructed with under 40 points) discarded. In addition, the user can define a threshold for the number of polygons retained (i.e. if the threshold is defined as 4, then the 4 largest polygons are retained). There is the additional option to manually verify the detected areas after these steps. This can be defined in the ~~*calcAutoExtents*~~*Area* ~~and~~ object's *calcAutoAreas* ~~functions~~ function with the boolean variable *verify*. This calls on the *Area* object's *verifyExtents* function~~in PyTrx~~, which cycles through all the detected ~~extents~~ area features in all the images and allows the user to manually verify each one based on a click-by-click basis. This can be a time-consuming process with long image sequences, but ensures that falsely-detected areas are discarded.

~~Using the *calcAutoExtents* function, the detected areas are returned as a set of coordinates for each polygon in the image plane. The *calcAutoAreas* function returns the detected areas as real-world coordinates.~~

3.6.2 Manual detection

~~Target areas and lines~~ Area and line features can be defined manually in the ~~coordinate plane of each image in a sequence~~image plane. This requires the user to click around the area of interest, which creates a set of points from which a polygon/line object is formed. The user input is facilitated by the *ginput* plotting function, which is available in ~~the *matplotlib.pyplot* package~~Matplotlib. The polygon/line object can subsequently be georectified to create an object with real-world coordinates. At present, the manual detection functionality allows the user to define one area/line within a given image plane.

An example of PyTrx's manual definition of line features is displayed in Fig. 8. Sequential terminus positions were defined within the image plane on a click-by-click basis, from which line objects were constructed and projected. Terminus profiles have been plotted between ~~20 August and 05~~ the 20th August and 5th September 2015 (every five days), providing a detailed record of changes in terminus position over time. This shows a gradual retreat in terminus position over a peak period in the melt season.

Currently, lines are limited to being manually defined, and only one line can be defined within a given image plane. Automated line detection would be a valuable addition in the future for detecting terminus profiles in sequential imagery of calving glacier fronts. However, attempts to detect terminus profiles from oblique time-lapse imagery has proved problematic due to reflections from the adjacent fjord water, ~~changes in tide~~ tide fluctuation, and changes in lighting and shadowing. With future development, it is hoped that these limitations can be overcome.

3.7 Image registration and georectification

~~*CamEnv* (found in~~ The CamEnv.py ~~) compiles and~~ script handles all information concerning the camera environment~~. This includes information concerning the camera calibration, which uses a typical pinhole camera model, and radial and tangential~~

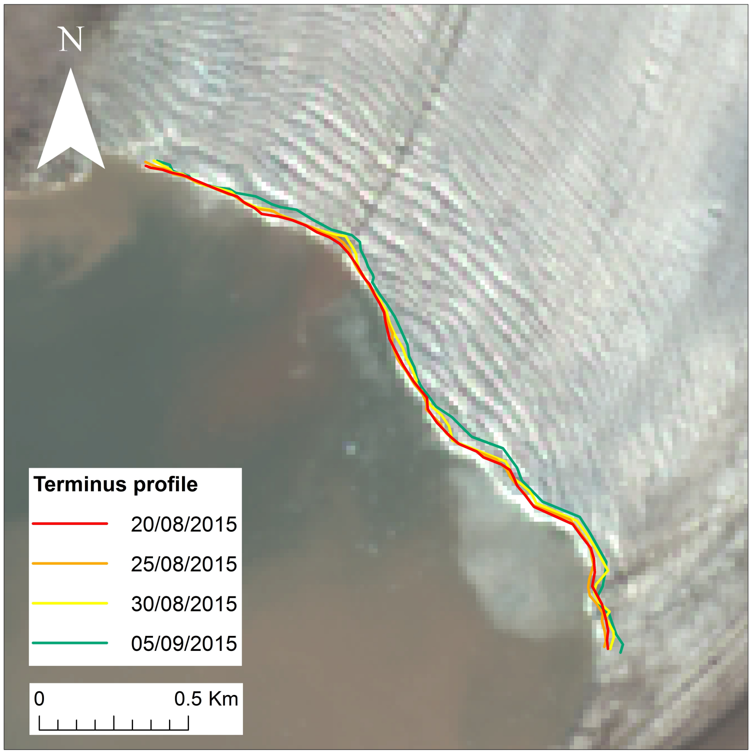

Figure 8. An example of PyTrx's ability to the extract sequential terminus profiles from Tunabreen camera T1. A similar example of this is provided with PyTrx at https://github.com/PennyHow/PyTrx.

~~distortion parameters. It also computes the homography from this along with the GCPs and DEM, using a planar projective transformation approach similar to ImGRAFT.~~ , including functionality for determining the intrinsic camera matrix and lens distortion coefficients either from a text file of raw information or a set of calibration images. Calibration using inputted chessboard images is carried out in the *calibrateImages* function, based on the approach available in the OpenCV toolbox. Camera calibration is automatically conducted during the initialisation of the *CamEnv* object when the input directory is defined as a folder containing a set of calibration chessboard images.

The corners of the chessboard are first detected in each image (using OpenCV's *findChessboardCorners* algorithm), based on the inputted chessboard corner dimensions which are defined by the user. If all corners of the chessboard are found, the locations of these corners are then defined in the image plane to sub-pixel accuracy (using OpenCV's *drawChessboardCorners* and *cornerSubPix* algorithms). Image plane coordinates for the detected chessboard corners from all of the images are subsequently used to calculate the intrinsic camera matrix and lens distortion coefficients (using OpenCV's *calibrateCamera* algorithm). A rough camera matrix and distortion coefficients is initially computed using the raw inputted coordinates. These are then optimised with a second calibration run, whereby the principal point (calculated initially) remains fixed. By doing this, the camera matrix and distortion coefficients are refined, reducing the errors. PyTrx returns the intrinsic camera matrix (as shown in Equation 5), the lens disortion coefficients $(k_1, k_2, p_1, p_2, k_3)$, and the calibration error estimate, which are used subsequently in PyTrx's image correction and georectification processes.

For image registration, the planar projective transformation encapsulates the three-dimensional rotation of the camera platform as movement around its horizontal, vertical and optic axes. The output rotations are used to correct false motion from feature track measurements. Rotation that cannot be accounted for from the two-dimensional displacements is returned as a root-mean-square (RMS) residual pixel value, and this RMS is the main measure of error (i.e. the 'noise'). This is returned along with the velocity (i.e. the 'signal') as a signal-to-noise ratio.

The georectification method follows a similar workflow to ~~ImGRAFT~~ the ImGRAFT toolbox (Messerli and Grinsted, 2015). The homography model is calculated based on a camera model (i.e. extrinsic and intrinsic parameters, and distortion coefficients), ~~for which information can be inputted to PyTrx via a . txt or .mat file format. This information is~~ which is used to compute the inverse projection variables. These variables are either defined by the user in the stand-alone functions, or compiled and stored in the *CamEnv* object and subsequently called upon by the *Velocity*, *Area*, and ~~xy points from either the velocity, area or line measurements are subsequently projected onto the DEM in order to obtain their corresponding three-dimensional coordinates~~*Line* objects.

An example of PyTrx's georectification capabilities, using images from Tunabreen (camera site 1, Fig. 3C), is shown in Fig. 9. Point locations in Fig. 9A denote the position of observed calving events (i.e. the break-off of ice from the glacier terminus) in the image plane. The colour of each point denotes the style of calving, ranging from small break-offs (i.e. waterline and ice fall events) to large collapses (i.e. sheet and stack collapses), and detachments that occur below the waterline (i.e. subaqueous events). These xy point locations have been translated to real-world coordinates using the georectification functions available in PyTrx (Fig. 9B).

The point locations are overlain onto a satellite scene of the glacier terminus, which was captured as close as possible to the time of ~~image capture~~ the time-lapse image acquisition (Fig. 9B). The point locations tightly follow the terminus position, demonstrating good accuracy in the georectification technique and the given information about the camera environment. However, points tend to deviate from the terminus position on the eastern side of the terminus, which is the furthest away from the camera. Similar deviation is evident in Fig. 8 also. This may indicate a degree of distance decay that is difficult to correct in the homography model. Distance decay is evident in other georectification methods, especially when performing georectification from monoscopic set-ups (James et al., 2016).

4 Evaluation of PyTrx

PyTrx and its modular object-oriented design make it an accessible toolbox for deriving measurements from oblique imagery. PyTrx is flexible and can be adapted easily for the user's requirements, as it is distributed as a set of files with simple example drivers. It broadens the range of photogrammetry toolboxes that are publicly available, with a Python-alternative to those coded in Matlab (Messerli and Grinsted, 2015; James et al., 2016) and C++ (Schwalbe and Maas., 2017).

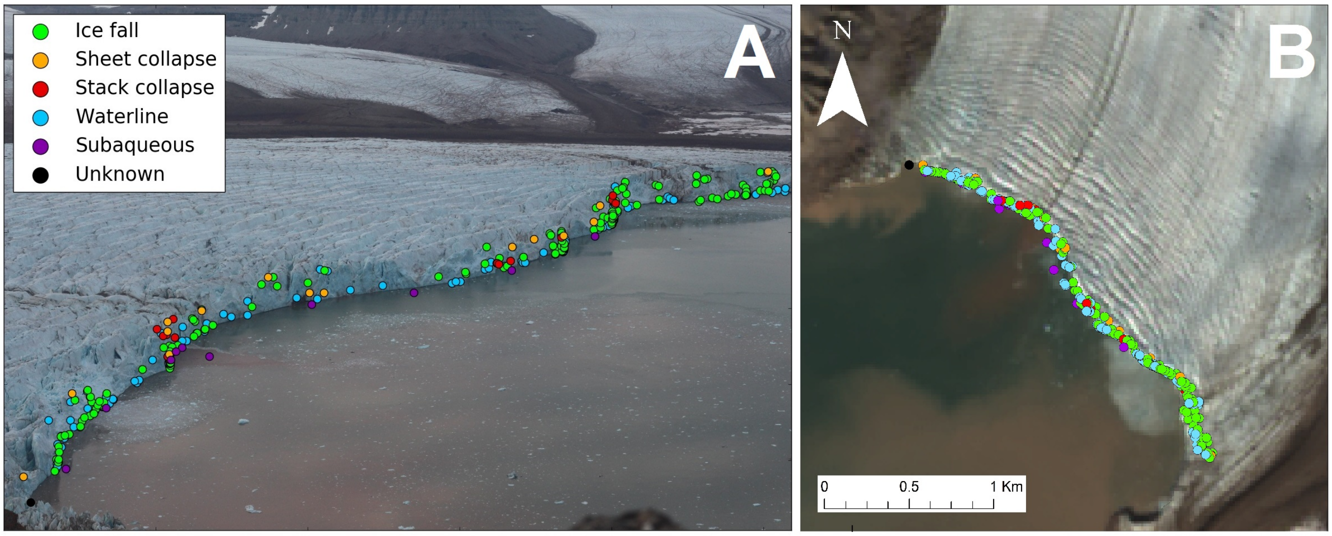

Figure 9. Calving events observed in the image plane (A) and georectified (B), with the colour of the point denoting the style of calving. Events were manually detected, from which the style of calving was interpretted. The time-lapse image is taken from a time-lapse sequence captured between the 7th and 8th August 2015. Figure adapted from How et al. (In Press). This example is provided with PyTrx at https://github.com/PennyHow/PyTrx.

The examples shown throughout demonstrate PyTrx's capabilities and range of applications in glaciology. Velocities are derived using an alternative approach, utilising an Optical Flow approximation that proves effective and processing-efficient. The addition of areal and line measurements to PyTrx's outputs have proved valuable in deriving surface areas of supraglacial lakes to show drainage events, and terminus profiles to examine glacial retreat.

The applications presented here also highlight functions to develop in subsequent releases of PyTrx. ~~The~~ For example, the detection method for areas ~~proves to be a robust method. However, it was~~ proved challenging to transfer ~~these~~ to achieve automated line detection ~~from oblique images~~ (as alluded to in Section 3.6). This functionality would be useful as an efficient approach to defining terminus positions from oblique time-lapse imagery. It would also be especially valuable for other glaciological applications, such as detecting grounding line features in satellite imagery~~(e.g., ?)~~.

~~Distance decay in the georectification method is visible in some of the examples shown, specifically where measurements span the entire planeof the image (e.g. figures 8 and 9B) . It is possible to limit this with accurate information about the camera environment (e.g. camera location, camera pose) and accurate GCPs that cover the entire image plane. However~~

Errors have been constrained and estimated using the examples presented previously, as summarised in Table 1. These errors are divided into pixel error that are introduced during the measurements in the image plane, and those associated with the transformation of the measurements into three-dimensional space (i.e. georectification) (Schwalbe and Maas., 2017) . Homography uncertainty is defined as motion in the camera platform that is not resolved by the homography model. In all cases, this is ~~often not possible as glaciers are challenging environments to conduct photogrammetric measurements. An~~

Table 1. Error estimations for deriving velocity, area and line measurements using PyTrx

Error source	**Average velocity error**	**Average area error**	**Average line error**
Homography uncertainty (px)	0.5111	0.1294	0.9863
Pixel tracking (px)	0.9667	–	–
Feature detection (px)	–	86.6870	18.8170
Total pixel error (px)	1.4778	86.8164	19.8033
Total 3D error (%)	±0.638	±0.638	±0.638

~~alternative is to explore other methods of projective transformations that are more suited to such applications.~~ constrained to less than one pixel (Table 1).

~~Error in PyTrx's feature-tracking approach is largely constrained due to its~~ When deriving velocities, pixel tracking error is defined as the difference between the original seeded point and the back-tracked point from the destination image to the reference image (as outlined previously in Section 3.5). This error can be adequately constrained using the back-tracking ~~verification algorithm for removing falsely-tracked points. Therefore error estimation is calculated simply as the signal-to-noise ratio; the signal being the tracked displacement, and the noise being the RMS value after image registration.~~ threshold, which is defined by the user and effectively removes point tracks that hold uncertainty.

Errors from area and line feature detection were determined based on sensitivity testing, which vary significantly based on the feature (as demonstrated in Table 1 and therefore it is advised to perform this sensitivity test when carrying out this approach in PyTrx (e.g., How et al., 2017). Toolboxes such as ImGRAFT and Pointcatcher use the Monte Carlo method, which is a more robust approach to determining error. The Monte Carlo method uses random repeated sampling to simulate variation in a system, and has been used in time-lapse photogrammetry to indicate the sensitivity of the image registration to the static point displacements (Messerli and Grinsted, 2015; James et al., 2016). PyTrx could benefit from well-constrained error analysis ~~also. In addition, error analysis for areal and line measurements would be valuable also, which are currently determined using sensitivity analysis (e.g., How et al., 2017)~~in future releases.

Errors from the georectification process have previously been defined using broad estimates (e.g., Messerli and Grinsted, 2015) or distance-based approximations (e.g., Schwalbe and Maas., 2017). A broad error estimate of ±0.638% was determined here through ground-truthing between the outputs from PyTrx and satellite images from the same acquisition period. Similar to Schwalbe and Maas. (2017), we found that the average error increases with baseline distance, and conservatively estimate an error of 0.2% up to a baseline of 1500 m, and 0.8% for a baseline of 1500–3000 m. For instance, the terminus lines depicted in Figure 8) have an average error of ±2.4 m over a baseline of 1500–2000 m, but this error grows to an average of ±7.7 m beyond a baseline of 2000 m. From 3000 m, the error increases exponentially and is difficult to adequately constrain given that the camera set-ups presented in this work do not cover further than 3500 m. This error is often difficult to adequately constrain due to the challenging environments that glacial photogrammetry studies are conducted in, and therefore an alternative in the future would be implement other methods of projective transformations that are more suited to such applications.

The glacial photogrammetry toolboxes currently available have aspects and functionality that make them unique and beneficial to use. For instance, ImGRAFT contains sophisticated functions to refine specified components of the camera environment (i.e. camera pose, location, and GCPs) to produce accurate projections (Messerli and Grinsted, 2015). Equally, multiple DEMs can be inputted into Pointcatcher to derive well-constrained vertical and horizontal displacements; even from challenging set-ups where image capture is not perpendicular to ice flow (James et al., 2016). The functions for deriving areal and line measurements (and their programming in Python) are what make PyTrx a unique toolbox. Users therefore have a greater range of toolboxes to choose from when embarking on glacial photogrammetry.

5 Conclusions

The PyTrx toolbox has been presented here to showcase its abilities in obtaining velocity, areal and line measurements from oblique time-lapse imagery. Images were collected using time-lapse cameras installed at Kronebreen and Tunabreen, two tidewater glaciers in Svalbard, from which the functionality of PyTrx could be tested.

The examples shown throughout demonstrate PyTrx's specific applications in deriving ice flow velocities, surface areas of supraglacial lakes and meltwater plumes, georectified point locations denoting the position of calving events, and glacier terminus profiles. PyTrx serves as a Python-alternative to the monoscopic toolboxes that are currently available, thus widening the applications of terrestrial photogrammetry in glaciology. Future development of the toolbox has been highlighted, which indicate promise in its growth and applications.

Code and data availability. The PyTrx toolbox is available on GitHub at https://github.com/PennyHow/PyTrx (715 MB). This includes driver scripts for a selection of the examples given in this paper. Datasets and example driver scripts for the specified examples presented in this paper are also available on this repository.

Author contributions. PH developed the PyTrx algorithms for obtaining geometric measurements (i.e. areas and lines), and revised the pre-existing functions to produce a coherent, fully documented toolbox. In addition, PH produced all example applications of PyTrx presented in this paper. NRJH developed the homography and georectification functions in PyTrx and supervised LB (née Addison) in developing the image handling and feature-tracking functionality. LB produced the initial skeleton of PyTrx and its work flow as part of her M.Sc. dissertation in Geographical Information Science, at the University of Edinburgh (2014/15).

Competing interests. There are no competing interests present.

Acknowledgements. This work is ~~affliated~~ affiliated with the CRIOS project (Calving Rates and Impact On Sea Level), which was supported by the Conoco Phillips-Lundin Northern Area Program. PH is funded by a NERC PhD studentship (reference number 1396698). The time-lapse camera enclosures used in this study were designed and manufactured by Alex Hart and the GeoSciences Mechanical Workshop at the University of Edinburgh. The DEM of Kongsfjorden originates from the Norwegian Polar Institute, kindly distributed in a modified form with permission, licensed under the Creative Commons Attribution 4.0 International (CC BY 4.0) license. The DEM of Tempelfjorden originates from ArcticDEM, which 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. We would like to thank the associate editor Jean Dumoulin and the two reviewers for their constructive feedback.

References

Addison, L., 2015. PyTrx: feature tracking software for automated production of glacier velocity. M.Sc. Thesis, University of Edinburgh, UK, 134 pp, URL: https://www.era.lib.ed.ac.uk/handle/1842/11794.

Ahn, Y. and Box, A., 2010. Glacier velocities from time-lapse photos: Technique development and first results from the Extreme Ice Survey (EIS) in Greenland. Journal of Glaciology, 56(198), 723–734, doi:10.3189/002214310793146313.

~~Baker, S. , Scharstein, D., Lewis, J. P., Roth, S. , Black, M. J. and Szeliski, R. ,~~

Ahn, Y. and Howat, I., 2011. ~~A database and evaluation methodology for Optical Flow. International Journal of Computer Vision 92~~Efficient Automated Glacier Surface Velocity Measurement From Repeat Images Using Multi-Image/Multichip and Null Exclusion Feature Tracking. IEEE Transactions on Geoscience and Remote Sensing **49**(80), 2838–2846, 10.1109/TGRS.2011.2114891.

Akcay, O. and Avsar, E. O., 2017. The effect of image enhancement methods during feature detection and matching of thermal images. The International Archive of the Photogrammetry, Remote Sensing and Spatial Information Sciences 42(1), ~~1–31,~~ 575–578, 10.5194/isprs-archives-XLII-1-W1-575-2017.

~~Busch, D. D.~~

Acharya, T. and Ray, A. J., 2005. Image Processing: Principles and Applications. Wiley, ~~2014. Digital SLR Cameras and Photography For Dummies, 5th edn. John Wiley& Sons, Hoboken,~~ New Jersey, ~~352~~ 451 pp.

~~Christie, F.D. W., Bingham, R. G., Gourmelen, N., Tett,~~

Baker, S.~~F. B. and Muto, A., 2016. Four-decade record of pervasive grounding line retreat along the Bellinghausen margin of West Antarctica. Geophysical Research Letters 43(11), 2016GL068972,~~ , Scharstein, D., Lewis, J. P., Roth, S., Black, M. J. and Szeliski, R., 2011. A database and evaluation methodology for Optical Flow. International Journal of Computer Vision 92(1), 1–31, doi:10.1007/s11263-010-0390-2.

Corripio, J. G., 2004. Snow surface albedo estimation using terrestrial photography. International Journal of Remote Sensing 25(24), 5705–5729, 10.1080/01431160410001709002.

Danielson, B. and Sharp, M., 2013. Development and application of a time-lapse photograph analysis method to investigate the link between tidewater glacier flow variation and supraglacial lake drainage events. Journal of Glaciology 59(214), 287–302, doi:10.3189/2013jog12j108.

Dietrich, R., Maas, H.-G., Baessler, M., Rülke, A., Richter, A., Schwalbe, E. and Westfeld, P., 2007. Jakobshavn Isbræ, West Greenland: Flow velocities and tidal interaction of the front area from 2004 field observations. Journal of Geophysical Research: Earth Surface 112(F3), F03S21, doi:10.1029/2006JF000601.

Eiken, T. and Sund, M., 2012. Photogrammetric methods applied to Svalbard glaciers: accuracies and challenges. Polar Research 31, 18671, doi:10.3402/polar.v31i0.18671.

Finsterwalder, R., 1954. Photogrammetry and Glacier Research with Special Reference to Glacier Retreat in the Eastern Alps. Journal of Glaciology 2(15), 306–315, doi:10.3189/S0022143000025119

Fox, A. J. and Nuttall, A. M., 1997. Photogrammetry as a research tool for Glaciology. The Photogrammetric Record 15(89), 725–737, doi:10.1029/2006JF000601.

Härer, S., Bernhardt, M. and Schulz, K., 2016. PRACTISE – Photo Rectification And ClassificaTIon SoftwarE (V.2.1). Geoscientific Model Development 9(1), 307–321, 10.5194/gmd-9-307-2016.

Harris, C. and Stephens, M., 1988. A combined corner and edge detector. Proceedings of the Alvey Vision Conference. 147–151.

Hartley, R. I. and Zisserman, A., 2004. Multiple View Geometry in Computer Vision, 2nd edn. Cambridge University Press, UK, 655 pp.

Heid, T. and Kääb A., 2012. Evaluation of existing image matching methods of deriving glacier surface displacements globally from optical satellite imagery. Remote Sensing of Environment 118(C), 339–355, doi:10.1016/j.rse.2011.11.024.

Heikkila, J. and Silven, O., 1997. A four-step camera calibration procedure with implicit image correction. Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1106–1112, doi:10.1109/CVPR.1997.609468

Horn, B. K. P. and Schunck, B. G., 1981. Determining Optical Flow. Artificial Intelligence. Artificial Intelligence 17(1–3), 185–203. 10.1016/0004-3702(81)90024-2

How, P., 2013. Measuring glacier movement and its influences using a new approach in terrestrial time-lapse techniques. M.Sc. Thesis. Lancaster University, Lancaster, UK. 142 pp.

How, P., Benn, D. I., Hulton, N. R. J., Hubbard, B., Luckman, A., Sevestre, H., van Pelt, W. J. J., Lindbäck, K., Kohler, J. and Boot, W., 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(6), 2691–2710, doi:10.5194/tc-11-2691-2017.

How, P., Schild, K. M., Benn, D. I., Noormets, R., Kirchner, N., Luckman, A., Vallot, D., Hulton, N. R. J., and Borstad, C., In ~~Review~~Press. Calving controlled by ~~melt-undercutting~~melt-under-cutting: detailed mechanisms revealed through time-lapse observations. Annals of Glaciology~~.~~, doi:10.1017/aog.2018.28

Huss, M., Sold, L., Hoelzle, M., Stokvis, M., Salzmann, N., Daniel, F. and Zemp, M., 2013. Towards remote monitoring of sub-seasonal glacier mass balance. Annals of Glaciology 54(63), 75–83, doi:10.3189/2013AoG63A427.

James, T. D., Murray, T., Selmes, N., Scharrer, K. and O'Leary, M., 2014. Buoyant flexure and basal crevassing in dynamic mass loss at Helheim Glacier. Nature Geoscience 7(8), 593–596, doi:10.1038/ngeo2204

James, M. R., How, P. and Wynn, P., 2016. Pointcatcher software: analysis of glacial time-lapse photography and integration with multitemporal digital elevation models. Journal of Glaciology 62(231), 159–169, doi:10.1017/jog.2016.27.

Jeong, S., Howat, I. M., and Ahn, Y., 2017. Improved multiple matching method for observing glacier motion with repeat image feature tracking. IEEE Transactions on Geoscience and Remote Sensing 55(4), 2431–2441, 10.1109/TGRS.2016.2643699.

Kääb, A. and Vollmer, M., 2000. Surface geometry, thickness changes and flow fields on creeping mountain permafrost: automatic extraction by digital image analysis. Permafrost and Periglacial Processes 11(4), 315–326, doi:10.1002/1099-1530(200012)11:4<315::AID-PPP365>3.0.CO;2-J.

Kääb, A., Lefauconnier, B. and Melvold, K., 2005. Flow field of Kronebreen, Svalbard, using repeated Landsat 7 and ASTER data. Annals of Glaciology 42(1), 7–13, doi:10.3189/172756405781812916.

Kalal, Z., Mikolajczyk, K. and Matas, J., 2010. Forward-Backward Error: Automatic Detection of Tracking Failures. 20th International Conference on Pattern Recognition, 2756–2759, doi:10.1109/ICPR.2010.675.

Kaufmann, V. and Ladstädter, R., 2008. Application of terrestrial photogrammetry for glacier monitoring in Alpine environments. International Archive of the Photogrammetry, Remote Sensing and Spatial Information Sciences, 37(B8), 813–818.

Kick, W., 1966. Long-term glacier variations measured by photogrammetry. A re-survey of Tunsbergdalsbreen after 24 years. Journal of Glaciology 6(43), 3–18, doi:10.3189/S002214300001902X.

Kraaijenbrink, P., Meijer, S. W., Shea, J. M., Pellicciotti, F., Jong, S. M. D. and Immerzeel, W. W., 2016. Seasonal surface velocities of a Himalayan glacier derived by automated correlation of unmanned aerial vehical imagery. Annals of Glaciology 57(71), 103–113, doi:10.3189/2016AoG71A072.

Lowe, D. G., 1999. Object recognition from local scale-invariant features. Proceedings of the Seventh IEEE International Conference on Computer Vision, Kerkyra, Greece, 1150-1157(2), 10.1109/ICCV.1999.790410

Leprince, S., Barbot, S., Ayoub, F. and Avouac, J. P., 2007. Automatic and precise orthorectification, coregistration, and subpixel correlation of satellite images, application to ground deformation measurements. IEEE Transactions on Geoscience and Remote Sensing 45(6), 1529–1558, 10.1109/TGRS.2006.888937.

Lucas, B. D. and Kanade, T., 1981. An iterative image registration technique with an application to stereo vision. Proceedings of the 7th International Joint Conference on Artificial Intelligence 2, 674–679.

Luckman, A., Benn, D.I., Cottier, F., Bevan, S., Nilsen, F. and Inall, M., 2015. Calving rates at tidewater glaciers vary strongly with ocean temperature. Nature Communications 6, 8566, doi:10.1038/ncomms9566.

Maas, H.-G., Dietrich, R., Schwalbe, E., Bässler, M. and Westfield, P. Analysis of the motion behaviour of Jakobshavn Isbrægletscher in Greenland by monocular image sequence analysis. The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences 36(5), 179–183.

Mallalieu, J., Carrivick, J. L., Quincey, D. J., Smith, M. W. and James, W. H. M., 2017. An integrated Structure-from-Motion and time-lapse technique for quantifying ice-margin dynamics. Journal of Glaciology, 1–13, doi:10.1017/jog.2017.48.

Medrzycka, D., Benn, D. I., Box, J. E., Copland, L. and Balog, J., 2016. Calving behavior at Rink Isbræ, West Greenland, from time-lapse photos. Arctic, Antarctic, and Alpine Research 48(2), 263–277, doi:10.1657/AAAR0015-059.

Messerli, A. and Grinsted, A., 2015. Image GeoRectification And Feature Tracking toolbox: ImGRAFT. Geoscientific Instrumentation Methods and Data Systems 4(1), 23–34, doi:10.5194/gi-4-23-2015.

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

Parajka, J., Haas, P., Kirnbauer, R., Jansa, J. and Blöschl, G., 2012. Potential of time-lapse photography of snow for hydrological purposes at the small catchment scale. Hydrological Processes 26(22), 3327–3337, doi:10.1002/hyp.8389.

Pętlicki, M., Ciepły, M., Jania, J. A., Promińska, A. and Kinnard, C., 2015. Calving of a tidewater glacier driven by melting at the waterline. Journal of Glaciology 61(229), 851–863, doi:10.3189/2015JoG15J062.

Rosenau, R., Schwalbe, E., Maas, H.-G., Baessler, M. and Dietrich, R., 2013. Grounding line migration and high-resolution calving dynamics of Jakobshavn Isbræ, West Greenland. Journal of Geophysical Research: Earth Surface 118(2), 382–395, doi:10.1029/2012JF002515.

Rosten, E. and Drummond, T., 2006. Machine learning for high-speed corner detection. European Conference on Computer Vision, Berlin, Heidelberg, 430–443, doi:10.1007/11744023_34.

Scambos, T. A., Dutkiewicz, M. J., Wilson, J. C. and Bindschadler, R. A., 1992. Application of image cross-correlation to the measurement of glacier velocity using satellite image data. Remote Sensing of Environment 42(3), 177–186, 10.1016/0034-4257(92)90101-O.

Schwalbe, E. and Maas, H.-G., 2017. The determination of high-resolution spatio-temporal glacier motion fields from time-lapse sequences. Earth Surface Dynamics 5, 861–879, 10.5194/esurf-5-861-2017.

Schild, K.M., Hawley, R.L. and Morriss, B.F., 2016. Subglacial hydrology at Rink Isbræ, West Greenland inferred from sediment plume appearance. Annals of Glaciology 57(72), 118–127, doi:10.1017/aog.2016.1.

Slater, D., Nienow, P., Sole, A., Cowton, T., Mottram, R., Langen, P. and Mair, D., 2017. Spatially distributed runoff at the grounding line of a Greenlandic tidewater glacier inferred from plume modelling. Journal of Glaciology 63(238), 309–323, doi:10.1017/jog.2016.139.

Shi, J. and Tomasi, C., 1994. Good features to track. Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 593–600, doi:10.1109/CVPR.1994.323794.

Soha, J. M. and Schwartz A. A. (1978) Multidimensional histogram normalization contrast enhancement. Proceedings of the 5th Canadian Symposium on Remote Sensing, 86–93.

Solem, J. E., 2012. Programming Computer Vision with Python: Tools and algorithms for analyzing images, 1st edn. O'Reilly Media, Sebastopol CA, 264 pp.

Szeliski, R., 2010. Computer Vision – Algorithms and Applications. Springer, London, 812 pp.

Tomasi, C. and Kanade, T., 1991. Detection and tracking of point features. International Journal of Computer Vision, Technical Report.

Vogel, C., Bauder, A. and Schindler, K., 2012. Optical Flow for glacier motion estimation. ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences 3, 359–364, doi:10.5194/isprsannals-I-3-359-2012.

Whitehead, K., Moorman, B. and Wainstein, P., 2014. Measuring daily surface elevation and velocity variations across a polythermal arctic glacier using ground-based photogrammetry. Journal of Glaciology 60(224), 1208–1220, doi:10.3189/2014JoG14J080.

Xu, G. and Zhang, Z., 1996. Epipolar geometry in stereo, motion and object recognition: a unified approach. Kluwer Academic Publishers, Dordrecht, The Netherlands, 316 pp., 10.1007/978-94-015-8668-9.

Zhao, F., Huang, Q. and Gao, W., 2006. Image matching by normalized cross-correlation. IEEE International Conference on Acoustics Speech and Signal Processing Proceedings, Toulouse, II-II. 10.1109/ICASSP.2006.1660446.

Zhang, G. and Chanson, H., 2018. Application of local optical flow methods to high-velocity free-surface flows: Validation and application to stepped chutes. Experimental Thermal and Fluid Science 90(C), 186–199, doi:10.1016/j.expthermflusci.2017.09.010.

Zhang, Z., 2000. A flexible new technique for camera calibration. IEEE Transactions on Pattern Analysis and Machine Intelligence 22(11), 1330–1334, doi:10.1109/34.888718.