Determine real-time glacier mass changes from camera images

Release 0.0.1

Joël Borner, Aaron Cremona

Dec 21, 2020

CONTENTS:

This is the official documentation.

CHAPTER

ONE

ALGORITHM 1: MATCHTEMPLATE WITH HISTOGRAMS

How to run:

1. Install dependencies and requirements
2. Import the script and run the main function

```
import mT_Hist
x, displacements, conversion_factors = mT_Hist.matchTemplate_hist("myfolder",
↪"template.jpg", 0.70, wait=1, vis=False, plotting=False, csv=True)
```

3. Analyze csv or plot the return values with pyplot.

1.1 Main Function

The workflow of the main function is as follows:

- Use the matchTemplate function.
- Remove duplicated matches
- Finding collinear matches
- Calculating displacements in pixel with the help of histograms
- Calculating a conversion factor from px to metric units with the help of histograms

mT_Hist.**matchTemplate_hist**(*folder_path*, *template_path*, *thresh*, *wait=1*, *vis=False*, *plotting=False*, *csv=True*)

Main function for Algorithm 1 (MatchTemplate with Histograms).

Parameters

- **folder_path** (*str*) – String with path to folder where the image series is located. can be an absolute or relative path
- **template_path** (*str*) – String with path to the image that should be used as template (image of one single tape)
- **thresh** (*float*) – Threshold for filtering matchTemplate. Recommended value is 0.7 - 0.75. See technical report for further info.
- **wait** (*int, optional*) – waiting time between 2 consecutive images if the calculation is visualized in [ms]. default is 1 ms.
- **vis** (*bool, optional*) – visualizes calculation (time-lapse with indicators). default is False

- **plotting** (*bool, optional*) – plots the calculation in real-time (parallel to vis). default is False. Note that real-time displacements are before post-processing, so smoothing of conversion factors with convolution is not applied yet! Requires PyQt5 library
- **csv** (*bool, optional*) – saves all output data into a csv file. default is True

Returns

- **x-axis** (*list of float*) – time values in hours for every image
- **total_displacement** (*list of float*) – total cumulative displacements for every image in cm. same length as x-axis
- **smooth_scales** (*list of float*) – distances between two tapes in px for every image. same length as x-axis

1.1.1 Important instructions for the template image

The template is an image of one tape. This template is then used to detect and track all tapes in the whole time series. Therefore, the chosen template can have a considerable impact on the results. Generally, the following criteria should be met to generate good results:

- The represented tape should have a high contrast to the background (no white, grey, yellow)
- The represented tape should have a high saturation (no white, black, grey, dark blue)
- The represented tape should be located in the middle of the pole, so the distortion differences are minimized.
- The template should contain a small margin on all edges
- The size of the represented tape must be the same as in the original image. It is recommended to create the template by cropping a full image, so there is no resizing.
- The represented tape should be in the center of the template image (no offset), so that underestimates resp. overestimates are reduced

Simply put, the template should be as representative as possible compared to all tapes, all images, all exposures. If the pole is always inclined for the whole time series, the tape on the template should be inclined as well. Ideally, the color of the tape in the template is red, because the contrast as well as the saturation is high.

Some examples for good templates:

Fig. 1: Examples for good templates. There is enough margin around the tapes, there is good saturation and contrast present.

The following images show some bad examples for templates:

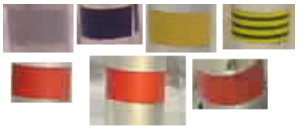

Fig. 2: Top row (left to right): low saturation/contrast, too dark and no saturation, good saturation but no contrast, texture filling. Bottom row (left to right): not enough margin, too much margin, distorted tape (too low on the pole)

1.1.2 Important instructions for the image series

The whole time series is a set of images in a folder. This folder is given the function as an input. The following criteria should be met:

- The filenames must be timestamps in the following format: yyyy-mm-dd_hh-mm (e.g. 2019-06-27_11-59.jpg) other image file types are also allowed.
- The algorithm is tested and optimized for time intervals of 20 minutes between images. Different intervals are generally possible (yet untested), but the displacements between two images must not exceed 4 cm.
- Footage from installation work at the beginning of the time series must be removed. The first image should only contain the installed pole.
- The initial position of the pole should preferably cover already the whole image height and not only the very bottom.
- Displacements during data gaps with more than 4 cm displacement will not be calculated correctly and have to be added manually.

1.1.3 Important instructions for the chosen threshold

The threshold filters results from the matchTemplate function comparing target image and template. Sub-areas of the target image with a correlation value higher than the threshold are considered a match and are used for further calculations. Some remarks:

- The algorithm has proven to work best for thresholds between 0.7 - 0.75
- It is recommended to run the program several times with different thresholds. This makes it very easy to detect outliers and calculation errors.
- Higher thresholds may lead to a loss of correlation and termination of the program if there is a longer period of bad quality images (e.g. snowfall)
- Lower thresholds may lead to higher instability of the algorithm (wrong displacement calculations)

1.2 Sub-functions

The following functions are called by the main function. Those functions are not intended to use individually. The documentation is therefore purely for a better understanding of the main function.

1.2.1 Removing Duplicates

mT_Hist.**remove_duplicates**(*points*)

This function is approximating points that are very close together into 1 single point. If 2 or more points are close together (defined by a threshhold in px), the average x and y coordinates of all those points determine the result point.

Parameters **points** (*list of tuple of float*) – list of x and y coordinates e.g. [[x1, y1], [x2, y2], . . .]

Returns **points** – list of x and y coordinates of fewer points.

Return type list of tuple of float

1.2.2 Finding Collinear Matches

mT_Hist.**find_collinear**(*points*)

This function searches for points that are collinear (on 1 straight line). If there are several lines, the one with the most points on it is chosen.

Parameters **points** (*list of tuple of float*) – list of x and y coordinates e.g. [(x1, y1), (x2, y2), ...]

Returns

- **collinear_points** (*list of tuple of float*) – list of coordinates of all matches that are collinear (on a straight line)
- **angle** (*float*) – the inclination of the pole. returns 0 if no collinear matches.

1.2.3 Calculating Displacements

mT_Hist.**get_distance**(*newmatches*, *oldmatches*, *angle*)

Detects the most common distance between 2 sets of points. This function is beneficial because all tapes on the pole are supposed to move the same amount of distance between 2 frames. Therefore, if as input all matches in the old frame and all matches in the new frame are chosen, the most common distance between all combination of points will be the displacement of the pole. To be more precise, the distance projected to the pole is calculated (hence the pole inclination as input).

Parameters

- **newmatches** (*list of tuple of float*) – List of point coordinates (x, y)
- **oldmatches** (*list of tuple of float*) – List of point coordinates (x, y)
- **angle** (*float*) – The inclination of the pole (retrieved with the find_collinear function)

Returns

- **bin_lower** (*float*) – lower limit of where the most common displacements were found.
- **bin_upper** (*float*) – upper limit of where the most common displacements were found.

mT_Hist.**compare_matches**(*matches*, *inclination*, *delta*)

This function compares the last two sets of points in a whole time series of points and calculates the displacement between those. If no displacement can be calculated, it recursively tries to compare earlier points with the current one.

Parameters

- **matches** (*list of tuple of float*) – A 2d list containing several points for each time step. e.g. [[[x11, y11], [x12, y12]], [[x21, y21], [x22, y22]]]
- **inclination** (*float*) – Inclination angle of the pole in rad
- **delta** (*int*) – difference between the positions of the images that are compared. default is 1 (last image compared to second last). delta is used to recursively iterate through past images to find good matches.

Returns

- **displacement** (*float*) – displacement between the frames in px.
- **delta** (*int*) – end point of the recursive iteration. shows to which frame the last frame was compared to. Example: If the last frame is compared to the second last delta = 1. delta is used to calculate the total displacement in a later step of the algorithm. The information is

needed, so that the found displacement of the current time step can be added to the correct value of the cumulative displacement.

1.2.4 Calculating Conversion Factor

mT_Hist.**get_scale**(*points*, *scale_array*)

Gets the most common distance between all matches in 1 frame, Which corresponds to the distance between 2 tapes.

Parameters

- **points** (*list of tuple of float*) – list of all matches in a frame
- **scale_array** (*list of float*) – list of float numbers of scales from previous frames, so an average can be calculated

Returns **scale** – the distance between 2 tapes in px.

Return type float

mT_Hist.**clean_scales**(*daily_disp*, *conversion_factors*, *boxlength*)

This function makes the calculated conversion factors smoother by applying a convolution function (floating average). This way, outliers (errors) are removed. From the smoothened conversion factors and displacement rates, a new cumulative displacement array is calculated.

Parameters

- **daily_disp** (*list of tuple of float*) – list of displacement rates between frames. The first value is the displacement in px, the second value is the relative index to wich image the current one was compared with (this is needed for calculation of cumulative displacements) Example: if the displacements of the current frame was calculated by comparing the current frame with the previous one, the index is 1. If the previous frame did not contain any matches and the displacement was calculated by comparing the current frame with one frame earlier than the previous the index is 2.
- **conversion_factors** (*list of float*) – list of calculated distances between two tapes from every frame. Same length as daily_disp
- **boxlength** – amount of values that should influence the floating average. Must be smaller than half of the length of conversion_factors.

Returns

- **clean_displacements** (*list of float*) – list of cumulative displacements between frames in cm with smoothened conversion factor
- **scale_smooth** (*list of float*) – smoothened conversion factors after convolution

mT_Hist.**px_to_cm**(*px*, *ref_cm*, *ref_px*)

Converts a number from px to cm according to a defined scale.

Parameters

- **px** (*float*) – distance in px that is to be converted.
- **ref_cm** (*float*) – reference distance in cm (e.g. distance between 2 tape stripes).
- **ref_px** (*float*) – reference distance in px (same reference as ref_cm)

Returns **distance** – converted distance in cm.

Return type float

1.2.5 Other functions

mT_Hist.**load_images_from_folder**(*folder*)
function loading all image files inside a specified folder. File names must contain date and time with delimiters - or _

Parameters **folder** (*string*) – path of the folder (string). can be a relative or absolute path.

Returns

- **images** (*list*) – list of opencv images
- **times** (*list of float*) – list of time differences to the first image in hours. e.g. [0, 0.5, 1.0, ...]

mT_Hist.**draw_rectangle**(*image*, *points*, *w*, *h*, *color*, *thickness*)
Draws a rectangles for a set of points. This function is mainly for visualizing and debugging purposes.

Parameters

- **img** (*opencv_image*) – The image where the recangles should be drawn
- **points** (*list of tuple of float*) – set of x and y coordinates. e.g. [(200, 100), (120, 150), ...]
- **w** (*int*) – width of the rectangles
- **h** (*int*) – height of the rectangles
- **color** (*tuple of int*) – color tuple in BGR
- **thickness** (*int*) – thickness of the drawn rectangle borders

CHAPTER

TWO

ALGORITHM 2: MATCHTEMPLATE WITH MEANSHIFT

How to run:

1. Install dependencies and requirements
2. Select a set of images (at least 100 images) with good lighting condition distributed during the whole period, and copy them into one folder. This calibration set of images will be used to calculate an appropriate conversion factor (to convert displacements from pixel to meter)
3. Select a set of templates (about a dozen) as explained in https://rtgmc.readthedocs.io/en/latest/algorithm1.html#important-instructions-for-the-template-image and save them into one folder
4. Import the script and run the main script, i.e. mT_mS.py with following inputs:

- path: select the path to the folder with image time series
- path_cal: select the path to folder with calibration set of images (step 2 from above)
- path_template: select the path to the folder with templates (step 3 from above)

The workflow of the algorithm is as follow:

1. In the main script (mT_mS.py) the image time series is imported with the following function:

mT_mS.**load_good_images_from_folder**(*folder*)
Function loading the image series inside a specified folder. File names must contain date and time with delimiters - or _. Acquisition taken during night are filtered out with help of a darkness threshold.

Parameters **folder** (*string*) – Path of the folder (string). can be a relative or absolute path.

Returns

- **images** (*list*) – Time series of images.
- **times** (*list*) – List of time differences for every image to the first image in hours. e.g. [0, 0.5, 1.0, . . .]
- **hour** (*list*) – List containing the hour when the image was taken.

2. The parameters (a, b) of the function used to convert displacements into metric unit are calculated for the images in the calibration set (good lighting conditions) with the following function:

mT_mS.**find_conversion_factor**(*img*)
Calculate a (slope) and b (y-intercept) of the function that describes tape height in function of its position in the image (because of image distortion).

Parameters **img** (*opencv-image*) – Image containing the stake with tapes.

Returns

- **a** (*float*) – Slope of the function.

- **b** (*float*) – Y-intercept of the function (in px).

3. The combination of following functions allow to calculate displacements of the pole with tapes for the time series of images.

2.1 Main Functions

Considering two consecutive images, the match_template function finds the initial location of the tapes in the first image. The function meanShift is than able to track the tapes in the consecutive image. This combination is implemented in the function mS_different_frames. During the implementation of the algorithm, errors arising from a template that was not perfectly centered were observed. In addition, since the pole and therefore the tapes may tilt over time, it is possible that a tape is centered at the beginning of the time series but loses its centering over time, thus leading to erroneous results. To overcome this problem, the function mS_same_frame recalls match_template and meanShift on the same frame, thus correcting possible errors from template offsets.

mT.**match_template**(*im*, *temp*)

Finds areas of an image that match (are similar) to a template image.

Parameters

- **im** (*opencv-image*) – (Source image) The image in which we expect to find a match to the template image.
- **temp** (*opencv-image*) – (Template image) The image which will be compared to the source image.

Returns **collinearMatches** – The coordinates of the matching points found (Left uppermost corner of the ROI).

Return type list

mS.**mS_same_frame**(*images*, *times*, *a*, *b*, *template*, *h*, *w*)

This function recalls the matchTemplate and meanShift functions on the same frame to calculate eventual offsets of the templates (which will be later corrected).

Parameters

- **images** (*list*) – Time series of images.
- **times** (*list*) – List of time differences for every image to the first image in hours. e.g. [0, 0.5, 1.0, ...]
- **a** (*float*) – Slope of the function used to convert displacements into metric unit.
- **b** (*float*) – Y-intercept of the function used to convert displacements into metric unit.
- **template** (*opencv-image*) – Template used to identify the tapes.
- **h** (*int*) – Height of the track window.
- **w** (*int*) – Width of the track window

Returns **dy_list** – List of offset value between template and tape in each image.

Return type list

mS.**mS_different_frames**(*images*, *times*, *a*, *b*, *template*, *h*, *w*, *dy_cal*)

This function recalls the matchTemplate function in one frame (to find the initial location of the tapes) and meanShift to track tapes into the consecutive frame. Displacement of tapes between two frames is calculated and cumulated over the hole series.

Parameters

- **images** (*list*) – Time series of images.
- **times** (*list*) – List of time differences for every image to the first image in hours. e.g. [0, 0.5, 1.0, . . .]
- **a** (*float*) – Slope of the function used to convert displacements into metric unit.
- **b** (*float*) – Y-intercept of the function used to convert displacements into metric unit.
- **template** (*opencv-image*) – Template used to identify the tapes.
- **h** (*int*) – Height of the track window.
- **w** (*int*) – Width of the track window
- **dy_cal** (*list*) – Offsets of the templates.

Returns

- **dy_list** (*list*) – Cumulative displacements for every image in m.
- **std_list** (*list*) – Cumulative standard deviation for every image in m.
- **count_nomatches_notrack** (*int*) – Number of images for which the displacement could not be calculated (value of 0 is assigned).

2.2 Sub-functions

The following functions are recalled by the match_template function to remove duplicates matches and find collinear matches, i.e. on one straight line.

mT.**find_collinear**(*points*)

This function searches for points that are collinear (on 1 straight line). If there are several lines, the one with the most points on it is chosen.

Parameters **points** (*list of tuple of float*) – list of x and y coordinates e.g. [[x1, y1], [x2, y2], . . .]

Returns

- **collinear_points** (*list of tuple of float*) – list of coordinates of all matches that are collinear
- **angle** (*float*) – the inclination of the pole. returns 0 if no collinear matches.

mT.**remove_duplicates**(*points*)

This function is approximating points that are very close together into 1 single point

Parameters **points** (*list of tuple of float*) – list of x and y coordinates e.g. [[x1, y1], [x2, y2], . . .]

Returns **points** – list of x and y coordinates of fewer points.

Return type list of tuple of float

Since this algorithm works with colors to track tapes, the following functions are used to mask tape colors.

mask.**yellow**(*image*)

Creates a mask for yellow color.

Parameters **image** (*ndarray*) – Image of which the mask is wanted.

Returns **mask** – Mask for yellow color of the image.

Return type ndarray

mask.**red**(*image*)
Creates a mask for red color.

Parameters **image** (*ndarray*) – Image of which the mask is wanted.

Returns **mask** – Mask for red color of the image.

Return type ndarray

mask.**green**(*image*)
Creates a mask for green color.

Parameters **image** (*ndarray*) – Image of which the mask is wanted.

Returns **mask** – Mask for green color of the image.

Return type ndarray

mask.**blue**(*image*)
Creates a mask for blue color.

Parameters **image** (*ndarray*) – Image of which the mask is wanted.

Returns **mask** – Mask for blue color of the image.

Return type ndarray

mask.**black**(*image*)
Creates a mask for black color.

Parameters **image** (*ndarray*) – Image of which the mask is wanted.

Returns **mask** – Mask for black color of the image.

Return type ndarray

2.2.1 Recommendations

- Run the algorithm with at least 10 different templates (template is a sensitive variable)
- If possible build the stations in such a way that lighting condition are goods, i.e. colors are well recognizable and the contrast is not too high
- By comparing the results of different templates, as well as results of Algorithm 1, some erroneous results may be filtered out thus obtaining better performances

CHAPTER

THREE

KNOWN ISSUES AND IDEAS FOR FURTHER DEVELOPMENT

The following section describes some problems with the algorithms that have not been fully solved, as well as ideas for further development.

3.1 Data Gaps

Both algorithms are unable to calculate the correct displacements if longer periods of data are missing. The maximum displacement that can be measured between two consecutive images is 4 cm and corresponds to a complete phase shift of the tapes on the pole. The algorithms are however capable of detecting such data gaps and returning a warning, so that missing displacements can be added by hand. This could be solved in the future by detecting color sequences. If the the predefined color sequence is also implemented, displacements could even be backtracked for cases where tapes from the first image have already moved out of sight. There is already an inactive color detection function in the first algorithm, which however does not work reliably. It will be very challenging to implement reliable color detection because in some lighting conditions, the colors are not even detectable by human eye. Under no circumstances should this function interfere with the continuous displacement calculations, as these few, unavoidable errors in colour recognition reduce the amount of detected tapes and thus generally reduce the stability of both algorithms. Given that these data gaps occur very rarely and a manual correction is generally not a lot of effort and less error-prone, it must be asked whether such an effort is worthwhile.

3.2 Possible Improvements with Different Tape Colors

Some tapes are easier to detect than others. Especially white and gray tapes are problematic due to their low contrast to the background and no color saturation. Also tapes with texture filling (yellow with black stripes) are not ideal. Contrast and saturation are the most important characteristics the tapes should have. Therefore, the stability of both algorithms can be increased by replacing problematic colors with red, blue, yellow or green tapes. However, it must also be mentioned that this is a direct conflict with the time gap problem. If the number of unique colours in the sequence decreases, the possibilities to trace back displacements from time gaps also decrease.

3.3 Link to Glacier Mass Balance

Strictly speaking, the algorithms measure a change in height of the glacier surface compared to a fixed pole. In order to convert this into a glacier mass balance in [m w.e.], the displacements therefore have to be multiplied by a factor ρ_{ice}/ρ_{water}. This is further complicated when cases like station 1001 are considered, which are installed on a snow cover. The mass balance there could be approximated with a factor ρ_{snow}/ρ_{water}. This however suggests that only snow is melting during the time when a snow cover is existent, which is not entirely true (see report). Also, the exact snow density and the transition from snow melt to ice melt remain unsolved. In contrast to already existing snow cover during installation, short-term changes in the mass balance due to summer snowfall cannot be taken into account at the

moment because the camera does not move relative to the pole both when snow falls and when the newly deposited snow cover melts. Furthermore, the current sign convention suggests that glacier melt equals positive displacements. All values must therefore be inverted, as the common known sign convention suggests that melt results in a negative mass balance.

3.4 Continuous Integration and Delivery

Due to lack of time and knowledge, there was no possibility to implement proper test environments on gitLab. This was not very problematic for this work, as the development of two algorithms in general did not cause many conflicts. However, in further work, such an implementation could still prove useful.

3.5 Runtime and Performance

The main objective of this work was to prove the feasibility of such a workflow to automatically calculate mass balances. The runtime of the algorithms played a subordinate role. There is still room for improvement, as some functions, such as finding collinear matches, have a runtime of $O(n^2)$. For the time series obtained, this was absolutely unproblematic, but when much more data is added, it could play a role. Nevertheless, it must be said that this time saving potential is a fraction of the time gain of automation compared to hand measurements.

3.6 Changes in Time Intervals

Both algorithms were tested and optimised for time intervals of 20 minutes between two consecutive images, except at night. With a shorter interval, no problems should be expected. However, for longer time intervals, the chance of a loss of correlation increases. Under no circumstances should the time intervals be so long that the displacements between two images exceed 4 cm. Strictly speaking, 4 cm is already too much in case no tapes can be detected in an image.

CHAPTER

FOUR

REQUIREMENTS

```
numpy~=1.19.2
opencv-python~=4.4.0.42
imutils~=0.5.3
matplotlib~=3.3.2
pyqtgraph~=0.11.0
PyQt5~=5.15.1
pandas~=1.1.3
scikit-learn~=0.23.2
XlsxWriter~=1.3.7
```

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX