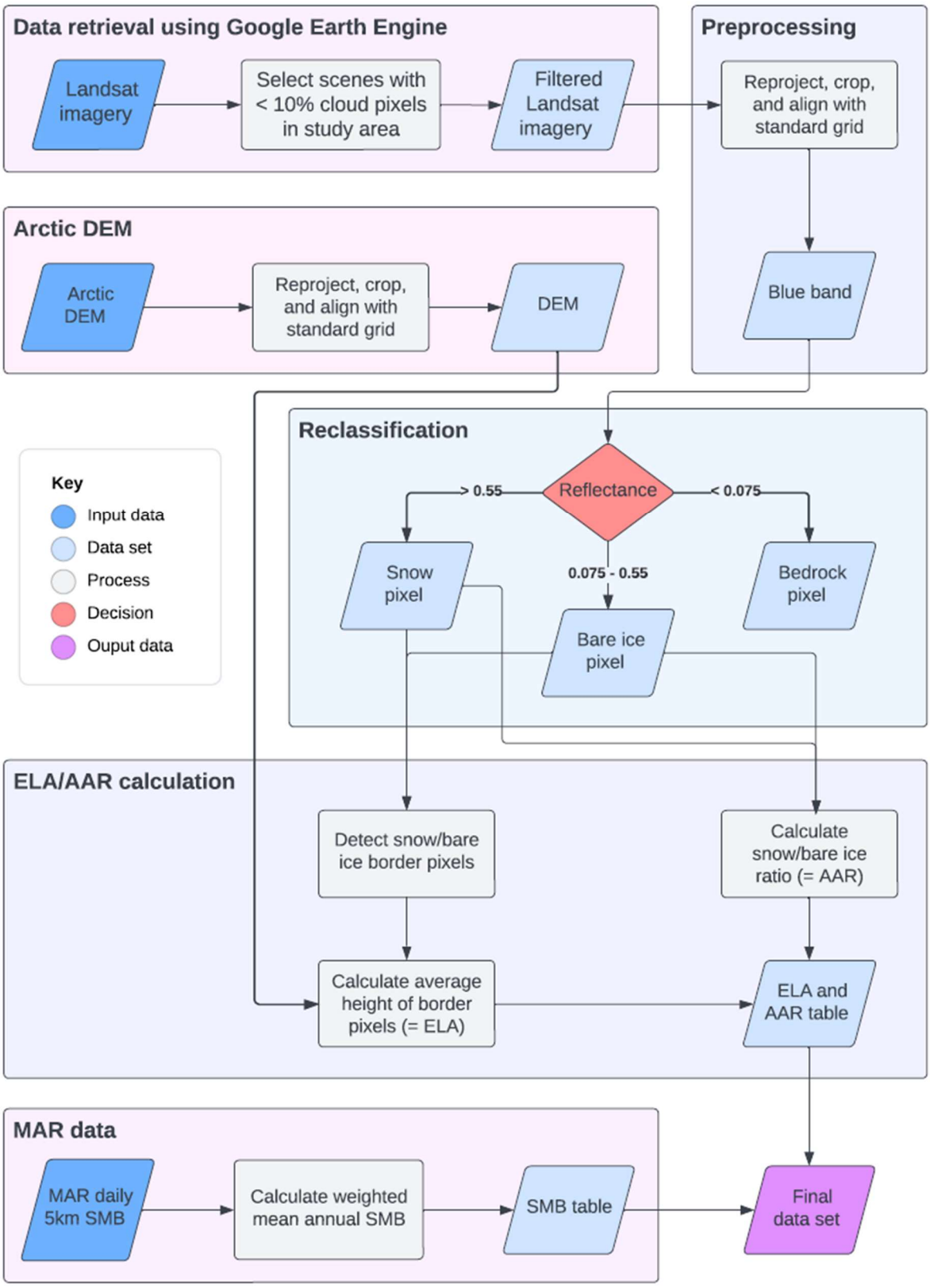
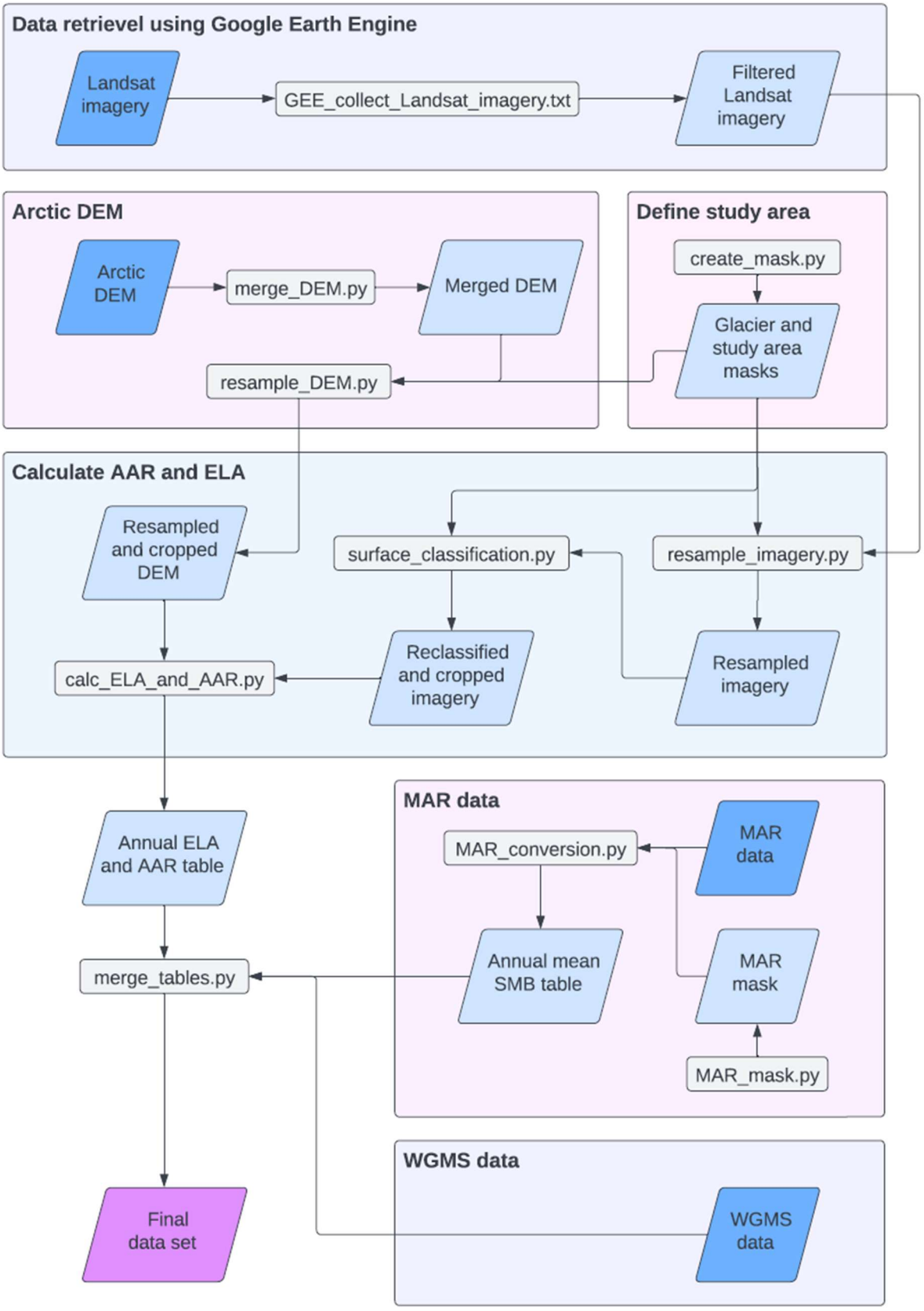
Code Documentation

Contents

Conceptual workflow



Script workflow



Create conda environment

- First create conda environment with all required packages by running the line below in the terminal (`requirements.txt` is located in the /Scripts folder)

```
conda create -n <name> -c conda-forge --file <path\to\requirements.txt>
```

Google Earth Engine

GEE_collect_Landsat_imagery.txt

- Google Earth Engine (GEE) script used to collect and export Landsat 5/7/8/9 imagery as GeoTIFFs with EPSG: 4326 – cloud masked and clipped to the extent of the study area
    - Since the `CLOUD_COVER` parameter for Landsat imagery only gives one value of the relative cloud cover of the entire scene (which is much bigger than most glaciers), this script uses the relative cloud cover over just the glacier as the basis for the export decision instead – this way the script collects the maximum number of usable dates and avoids cloudy images even when most of the scene is cloud-free
    - Use the script `GEE_collect_Landsat_imagery_simple_cloud_mask.txt` for a faster, more stable, but less thorough image retrieval
        - This script uses the CLOUD_COVER parameter for image filtering and the less accurate GEE cloud masking algorithm for masking
- Notes:
    - The calculation of the cloud mask and cloud ratio are fairly resource intensive (especially since GEE isn't really meant to be used with if-statements the way it is done here), so the code tends to crash often. If this happens reduce the number of images in the dataset, by either lowering the `CLOUD_COVER` threshold or reducing the date range
    - The script expects two different masks:
        - `mask_geometry`: shapefile outlining the glacier
        - `export_geometry`: shapefile outlining the study area
    - The cloudiness of a pixel is determined by the CFMask algorithm (https://www.usgs.gov/landsat-missions/cfmask-algorithm) and stored in the `QA_PIXEL` band of each Landsat scene
    - Currently only exports bands 1-5 + 7 since that is all that is needed, but more bands can be added
        - Note that some bands are stored in different data types, which might cause problems if not changed to one standard data type
    - Study area mask used in GEE should be at least a few pixels larger (in every direction) than the mask used in the analysis later to prevent issues during the reprojection/resampling
        - Easiest way is probably just to manually draw a rectangular mask in GEE and rename it to `export_geometry` as expected by the script
- Things that need to be adjusted for different glaciers/sensors:
    - Masks used to clip the image collection and calculate the cloud ratio (see `export_geometry` and `mask_geometry`)

DEM

merge_DEM.py

- Used to merge multiple raster tiles to a single file – in this case the ArcticDEM
- Notes:
  - Output resolution can be adjusted in the script, but will be changed in a later script during the warping process anyways (see `resample_DEM.py`)
- Things that need to be adjusted for different glaciers/sensors:
  - None (expect for paths, filenames, etc.)

resample_DEM.py

- Used to reproject/resample the DEM to the common grid used in the analysis (which is based on the grid from Landsat imagery obtained from the USGS Earth Explorer) and crop it to the extent of the glacier
  - Includes a different approach with the `rasterio` package that is not needed
- Notes:
  - After the reprojection, the output rasters will almost perfectly align with the common grid, but can still be off by a few centimeters – this is fixed with the adjustment of the GeoTransform
    - The DEM_reproj.tif raster will be deleted after the script ran through
  - There have been problems with `gdal.Warp()` being unable to perform the `cropToCutline` because of reprojection issues with the mask shapefile – should work fine now, but if you get an error along the lines of "`Invalid latitude/longtitude`" try using a mask shapefile projected to EPSG 4326, since that is the coordinate system that the GEE files are exported at
    - See `create_mask.py` to create both raster and shapefile masks in different coordinate systems
- Things that need to be adjusted for different glaciers/sensors:
  - Input mask, both the raster and shapefile
  - Destination coordinate system – see `dst_crs`
  - Pixel size/resolution – see `res`

Landsat analysis

create_mask.py

- Used to create a GeoTIFF to mask glacier extent and polygonise it as a shapefile in any given coordinate reference system
- Notes:
    - Make sure that the extent of the mask allows for an integer number of pixels at the desired resolution to make all raster data sets align exactly. Otherwise, there might be issues with edge pixels later on
    - Polygonization sometimes causes the kernel to crash, but you still get the right results out
- Things that need to be adjusted for different glaciers/sensors:
    - Coordinates of the mask raster
    - EPSG code of target CRS

resample_imagery.py

- Used to reproject the Landsat imagery obtained from Google Earth Engine to the common grid used in the analysis – same process as in `resample_DEM.py`
- Notes:
    - After the batch reprojection, the output GeoTIFFs will almost perfectly align with the common grid, but can still be off by a few centimeters – this is fixed with the batch adjustment of the GeoTransform
        - The GeoTIFFs created in the first step (found under ./Reprojected) can therefore be deleted – I have not yet found a good way to work around this extra export step
    - There have been problems with gdal.Warp() being unable to perform the cropToCutline because of the reprojection issues with the mask shapefile – should work fine, but you get an error along the lines of "`Invalid latitude/longtitude`" try using a mask shapefile projected to EPSG 4326, since that is the coordinate system that the GEE files are exported at
        - See `create_mask.py` to create both raster and shapefile masks in different coordinate systems
- Things that need to be adjusted for different glaciers/sensors:
    - Input mask, both the raster and shapefile
    - Destination coordinates system – see `dst_crs`
    - Pixel size/resolution – see `res`

surface_classification.py

- Used to crop the Landsat imagery to glacier extent, classify the glacier surface into bedrock/ice/snow pixels based on a simple threshold value classification applied to band 1 (blue), and detect border pixels for later ELA calculation
    - Includes a section at the start that serves to calculate the relative amount of no data-pixels, so that it is possible to filter images based on the ratio of no data- to data-pixels while differentiating between true no data and no data as the result of cropping the data to the extent of the study area

- Notes:
    - Classification could be replaced with a more sophisticated multi-band approach (e.g., random forest or neural network) later
    - Sensor defaults to maximum reflectivity in band 1 for Landsat 05 on snow pixels due to the high albedo which can be used to accurately classify the surface
    - Class legend:
        - 0 = no data/outside of glacier extent
        - 1 = bedrock
        - 2 = bare ice
        - 3 = snow
    - Border pixels consider k1-Moore-neighborhood and prioritize snow/ice-border over bedrock-border classification if multiple border types are possible
    - Border legend:
        - 4 = bedrock/bare ice border – meaning bedrock pixel that borders a bare ice pixel
        - 5 = bedrock/snow border
        - 6 = bare ice/snow border
        - 7 = bare ice/bedrock border
        - 8 = snow/bare ice border
        - 9 = snow/bedrock border
    - Output reclass.tif and border.tif files include only one band, but are still recognized as multiband in QGIS – not affecting the analysis, but still strange, as the script specifies to only export 1 band
- Things that need to be adjusted for different glaciers/sensors:
    - Input mask, both the raster and shapefile
    - Destination coordinates system – see `dst_crs`
    - Pixel size/resolution – see `res`
    - Threshold values for surface classification
        - Easiest way is probably basing them on a histogram of the original image
    - Threshold for acceptable amount of no data-pixels

calc_ELA_and_AAR.py

- Used to calculate the ELA and AAR for a given glacier over time. Outputs CSV table with ELA and AAR values for each input scene
- Notes:
    - The script can handle some special cases such as no snow or ice pixels, but does not account for cases where the input raster is only covering a part of the glacier
        - Best to visually inspect the surface classification raster to ensure reliable output data – this is issue is party addressed by the no data pixel threshold in `BATCH_surface_classification.py`
    - For ELA, the height of each snow/bare ice border pixel (value 8) is extracted from the DEM and then averaged
    - For the AAR the number of snow pixels is divided by the sum of snow and ice pixels
    - CSV table is saved both as `ELA_AAR_table_latest.csv` and `ELA_AAR_table_<current_date_and_time>.csv`

        - `ELA_AAR_table_latest.csv` is overwritten every time the script is run, so make sure to change file name in `BATCH_plotting.py` if you want to work with an older version
        - Just implemented this to avoid having to manually change the input file name in `BATCH_plotting.py` every time you run the script
- Things that need to be adjusted for different glaciers/sensors:
    - None (except for paths, filenames, etc.)

merge_tables.py

- Script that reads in the original ELA and AAR data, as well as the WGMS FoG data base and the MAR reanalysis data to create the final data base and export it as a CSV table
- Notes:
    - Calculates melt season length as the number of days between the days of lowest accumulation area extent for each year
    - CSV table is saved both as `complete_table_latest.csv` and `complete_table_<current_date_and_time>.csv`
        - `complete_table_latest.csv` is overwritten every time the script is run, so make sure to change file name in the statistical analysis scripts if you want to work with an older version
- Things that need to be adjusted for different glaciers/sensors:
    - Adjust first value in index list to the base date from which to calculate the first melt season length

MAR data

MAR_mask.py

- Creates a new mask to extract MAR data in GeoTIFF and shapefile format
- Notes:
    - MAR mask is larger than the normal study area masks to account for the much lower resolution of MAR (5km) to make sure that all pixels that lie within the normal study area are fully exported as they are cut away by GDAL if less than 50% of them lie within the extent of the normal study area
    - Size of MAR mask is determined by the extent of the normal study area mask and a buffer corresponding to the length of the diagonal of a MAR pixel (5km*sqrt(2))
        - Could get away with a mask half this size too if disk space is limited, but the files will be automatically deleted by `MAR_conversion.py`
- Things that need to be adjusted for different glacier/sensors:
    - Size and resolution of mask
    - Destination CRS

MAR_conversion.py

- Converts raw NetCDF files to GeoTIFF, reprojects them to the common grid of the study area, and crops them to the extent of the glacier. Afterwards the average daily SMB is exported as a CSV
- Things that need to be changed for different glaciers/sensors:
    - Size and resolution of raster datasets

Statistical analysis and plotting

correlation_analysis.py

- Analyses the statistical correlation between the original data and the data found in the WGMS FoG data base and creates a scatterplot for each correlation
- Notes:
    - First checks whether all data series in the input data are normally distributed using the Shapiro-Wilk test
    - Based on the normality distribution they correlation is assess using either Pearson's correlation coefficient (both data series normally distributed) or Kendall's tau (at least one data series not normally distributed)
        - An alternative for non-normal data would be Spearman's rho, which is supposedly better suited for small data sets, but as a strong correlation between the data series is expected, Kendall's tau is still thought to be the better option (use Spearman's rho by replacing `stats.kendalltau` with `stats.spearmanr`)
    - Scatterplots include a 1:1 relationship line as a visual aid to assess the direction of the correlation
- Things that need to be adjusted for different glaciers/sensors:
    - Depending on the outcome of the normality test, adjust the used correlation test in the script

regression_analysis.py

- Analyzes the relationship between AAR and SMB and calculates a linear regression to determine the equilibrium ratio AAR_0. The script then checks whether the regression is fulfilling the 4 assumptions of linear regressions:
    - Independence of residuals (using the Durbin-Watson test)
    - Homoscedasticity of residuals (using Breusch-Pagan test)
    - Residuals follow a normal distribution (using Shapiro-Wilk test)
    - Linear relationship between the variables is assessed through visual inspection of the residuals, the created trend plot, and the p-values of the coefficient and intercept of the linear model
- Notes:
    - Script analyzes the following relationships:
        - AAR over time
        - ELA over time
        - SMB over time
        - AAR vs. SMB
        - AAR vs. SMB (WGMS FoG data)
- Things that need to be adjusted for different glaciers/sensors:
    - Manually drop years without data, check for outliers etc.

moving_window_analysis.py

- Analyzes the evolution of AAR_0 over time through a 10-year moving window. Note that the size of the window is too small for the `sma.OLS` function to compute a realiable kurtosis score for the model.
- Notes:
    - Script investigates both the original data and the WGMS FoG data and creates a AAR_0 and R^2 over time plot
- Things that need to be adjusted for different glaciers/sensors:
    - Manually adjust lowest and highest AAR_0 time frames for plotting as well as induvial label positions