West University of Timisoara

Faculty of Mathematics and Computer Science

Study Program:
Computer Science in English

Master Dissertation

COORDINATOR:

Associate Prof. Marc Eduard Frîncu

GRADUATE:

Maria Minerva Vonica

Timişoara
2021

West University of Timişoara

Faculty of Mathematics and Computer Science

Study Program:
Computer Science in English

Master Dissertation

Name

COORDINATOR:

Associate Prof. Marc Eduard Frîncu

GRADUATE:

Maria-Minerva Vonica

Timişoara
2021

Abstract

Over the last decade, climate change has impacted Earths' atmosphere and environment more than anytime before. Studies held by NASA confirmed that the second hottest year ever recorded was 2019 and the recorded temperatures of the ocean have never been higher. One way to track these changes is to study the retreat of glaciers, since they are the most sensitive climate change indicators. Since NASA and USGS launched their first Earth observation satellite in 23 July 1972, millions of images of the Earth have been stored and are freely to use for various fields of research.

Having this in mind, the goal of our application is to analyse the retreat of glaciers over time by using advanced computer vision algorithms which are able to detect movement of pixels based on a time series of aerial scenes collected from the Landsat 8 satellite. By analysing the directions of glacier retreat over time, we try to generate future images of how they might change. The snowfall and ice percentages of a glacier for a satellite image are also taken into consideration as a validation of our results. The application has also been made with the goal of being simple to use and easy to access for anyone who wants to get involved and make a contribution.

This thesis contains information about the application's functionality, processing mechanics and experiments on various glaciers. In order to have a self sufficient environment, both search and download of Landsat 8 assets facilities have been made available, such that one does not need to have an already created dataset of glacier satellite images. As for generating a motion predicted image, we need to extract movement information between two consecutive snow index scenes which produces vectors of movement for each pixel. These results are used to further extend this movement trend which generates a predicted image.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The aim of Earth observation satellites is to collect satellite imagery in a consistent and long-term manner. These images can then further be used in order to extract information on weather behaviour, ocean temperatures, vegetation health, droughts and various other environmental variables. There are application which extract patterns based on these factors in order to understand why certain phenomena happened in the past and what are the conditions required for them to reappear in the future as well.

However, applications which track these changes require large processing power and consistent, qualitative datasets. Most of these datasets can be freely accessed and various analysation techniques can be used in order to extract pattern changes throughout the years. Fortunately, there are numerous Earth observing satellites which could satisfy the requirements of such a dataset. For the purpose of this thesis, only images collected by the Landsat 8 satellite have been used, since their quality is better than the last missions.

1.1 Motivation and goals

The motivation behind this thesis is found in the background of what climate change meant in the past and what it could further inquire in the future. Carbon dioxide levels in the atmosphere grown more than they can be naturally eliminated, causing butterfly effect changes in the environment. Glaciers are the most sensitive to these changes and through the means of Erath observation satellites they have been tracked longed enough such that qualitative imagery research can be used on them. On top of this, over the last years we have developed more and more powerful tools which are able to process heavy

amounts of data. Not taking advantage of these factors seems like a loss.

The goal of this thesis is to make use of such data by creating a tool for easy dataset creation, as well as analysing changes of glaciers in the last decade with the purpose of creating predicted images which could highlight future glacier retreat and snow fall decrease.

1.2 Related work

There are multiple research papers on the domain of glacier analysis. Some of those which we found useful for our research regarding analysis of glacier based on satellite imagery are:

- [WKN16], which proposes various methods of mapping glaciers on satellite images while taking into consideration seasonal variations in spectral properties of snow;
- [RRA19]: methods for separating ice and snow from debris using different indicators to calculate the mass balance of the glacier;
- [TK20]: assessment of glacier mass loss through a remote sensing modelling framework;
- [SHWS18]: tracking transient snow lines and investigating firn evolution;
- [MKI]: methods for measuring snow and glacier ice properties with the use of new satellite sensors, such as synthetic aperture radar.

For the alignment part of the thesis, some of the most helpful papers were [ERB], [VA] and [CMZ] which describe different ways in which we can pair the ORB alignment algorithm with different methods of warping and feature point extraction.

1.3 Our Contribution

This thesis makes use of techniques such as computer vision and machine learning for generating predicted images based on detected motion. This can be used to highlight glacier retreat over longer periods of time. For this purpose we had to implement techniques to align our images with respect to a reference, as well as determining ways to make use of the extracted optical flow information such that we can generate new images.

For the alignment process, more than 200 pairs of images have been tested, while only 10% of them having erroneous alignment results. As for the image generation, the results can be seen in the Section 4. TODO MORE TO WRITE HERE, SOME NUMBERS. Since the generated images rely mostly on visual validation, we have compared the ice and snow coverage on multiple datasets with different coordinates. TODO WRITE THIS BETTER.

The results were achieved through indexing glacier datasets by a crawler, which later on structure the data into path and row coordinates (see Paragraph 2.1.1), such that we are working with geographically consistent data. The normalized snow difference index has been calculated as described in Section 2.2.1, by using two Landsat 8 bands. We have chosen this metric to enhance snowfall and ice due to their high reflectance in the visible spectrum and high absorption in the infrared one.

Our goal is to process these images as pairs, so we had to take into consideration the fact that Landsat 8 does not have a perfectly stable trajectory, which yields in slightly misaligned pictures. However, this discrepancy would prove to generate unreliable results in the predicted image, therefore multiple techniques of solving this problem have been tested. Details of these can be found at Section 2.3.

For the image generation we used a computer vision algorithm which calculated the optical flow between two consecutive frames, which resulted in a matrix holding the distance of movement for each pixel. Based on this distance, new coordinates have been predicted for the pixels and they have been moved accordingly, resulting in a new image representing the change over time. Various methods of filtering have been applied on the results, as it is described in Section2.3. In order to validate our results, we have computed predicted images for all the pairs of the dataset, and we have compared them to already existing consecutive ones. The snow and ice coverage of the motion generated image can be then compared with its neighbours.

Since our application focuses on working with images, a graphical user interface seemed fit for this purpose. However, for the search and download part, a simple command line script has been used (see Section 2.4.

Chapter 2

Application

2.1 Dataset

In order to build the dataset, we made use of the freely available data from the Landsat Archive, specifically from collection 1, level 1. This dataset contains assets which are generated from the Landsat 8 Sensors, as well as entries from other older Landsat satellites. For the purpose of this paper, we will focus only on images collected from the Landsat 8 satellite.

2.1.1 Landsat 8

We will use images collected by the Landsat 8 satellite (Figure 2.1), which was the most recently lauched on the Atlas-V rocket (Vandenberg Air Force Base), in California on February 11, 2013. For remote sensing, Landsat 8 is equipped with two sensors, the Thermal Infrared Sensor (TIRS) and Operational Land Imager (OLI) one.

The satellite is orbiting the Earth at an altitude of 705 km, completing one orbit every 99 minutes. Based on this trajectory, the satellite has a 16 day repeat cycle and it acquires 740 scenes each day. These are organised on the path/row system defined by Worldwide Reference System-2 (WRS-2). A Landsat 8 scene size is 185 km x 180 km [LANa].

Worldwide Reference System-2 We will be referring to each scene's location based on its path and row coordinates from the worldwide reference system-2, which is a notation system used for Landsat images. This system is used with the main goal of keeping

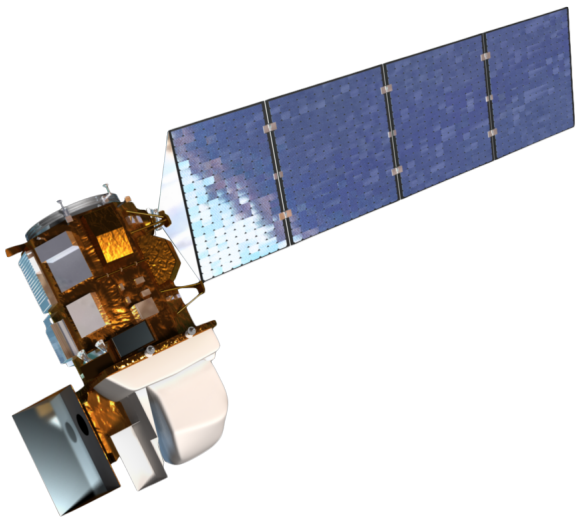

Figure 2.1: Landsat 8 satellite [LANb]

a structured archive of Landsat data which can be easily catalogued and accesses. Users can access aerial imagery through querying the specified path and row variables [wrs]. Landsat's trajectory projected on the world map can be seen in Figure 2.2.

Landsat scene naming convention

Each Landsat scene is named after a well-defined convention in order to easily check information such as the WRS path, row and the date of acquisition. Having access to this information without the need to download the scene itself or the metadata file which holds this information represents a valuable asset, since we can easily filter the data based on the naming convention itself. In the table 2.1 below we represent what each part of a Landsat scene of the form **LXS PPPRRR YYYYDDD GSIVV** means.

Operational Land Imager

The Operational Land Imager (OLI) represents remote sensing instrument which can be found abroad Landsat 8. It is built by Ball Aerospace & Technologies and it collects moderate resolution data which is used for monitoring changes in trends on the surface and evaluating how it changes over time. The images and data which OLI has helped collect have practical applications today in various fields such as mapping and monitoring changes in snow, ice, water and agriculture, [KK14]. The OLI operates in the short wave infrared spectral and visible region with a width of 185 km. Wavelengths of 443 nm to

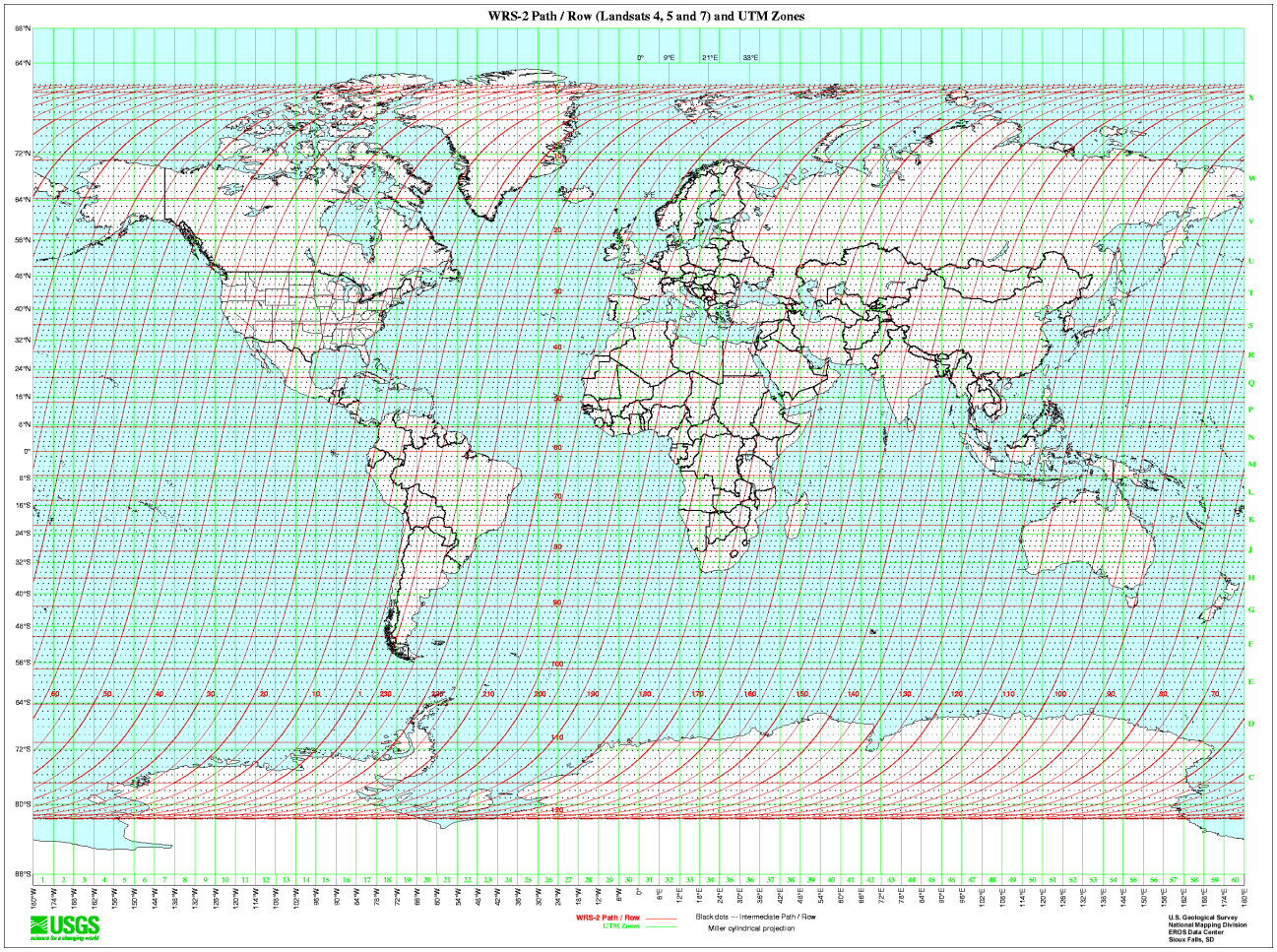

Figure 2.2: WRS-2 Path/Row for Landsat [wrs].

2,200 nm are translated into nine channels, from which eight are multispectral and one is panchromatic. The eight multispectral ones have a 30-meter spatial resolution, while the panchromatic channel has a 15 meters one. The OLI generates 9 bands for Landsat as shown in Figure 2.3.

Thermal Infrared Sensor

The Thermal Infrared Sensor (TIRS) measures land surface temperature in two thermal bands with a new technology that uses quantum mechanic techniques in order to detect thermal infrared wavelengths of light which are emitted by the Earth [lgn]. The thermal infrared sensor generates 2 bands for Landsat as shown in Figure 2.4.

L	Landsat
X	Sensor
SS	Satellite
PPP	WRS path
RRR	WRS row
YYYY	Acquisition year
DDD	Julian day of the acquisition year
GSI	Ground station identifier
VV	Archive version number

Table 2.1: Landsat 8 scene naming convention [sn].

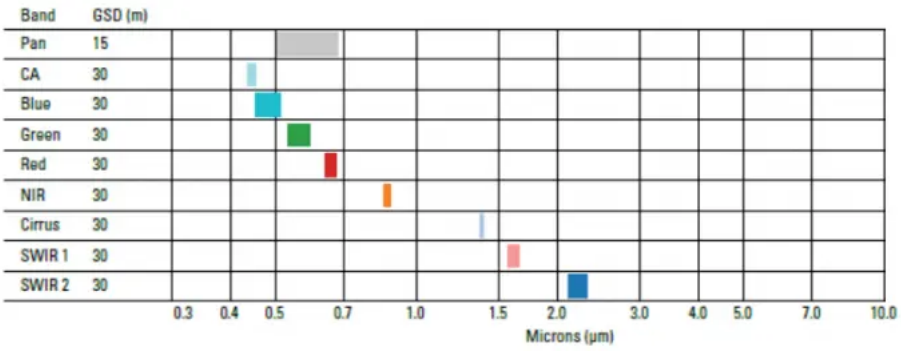

Figure 2.3: Landsat 8 OLI generated bands [l8o]

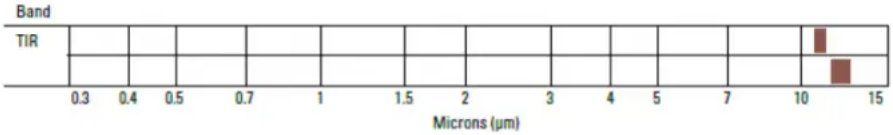

Figure 2.4: Landsat 8 TIRS generated bands [l8o]

OLI and TIRS bands

Landsat 8 acquires data from these sensors in in 11 bands, as following in Table 2.2.

Bands	**Wavelength (micrometers)**	**Resolution (meters)**
Band 1 (Coastal aerosol)	0.43-0.45	30
Band 2 (Blue)	0.45-0.51	30
Band 3 (Green)	0.53-0.59	30
Band 4 (Red)	0.64-0.67	30
Band 5 (Near Infrared)	0.85-0.88	30
Band 6 (SWIR1)	1.57-1.65	30
Band 7 (SWIR2)	2.11-2.29	30
Band 8 (Panchromatic)	0.50-0.68	15
Band 9 (Cirrus)	1.36-1.38	30
Band 10 (TIR 1)	10.6-11.19	100
Band 11 (TIR 2)	11.50-12.51	100

Table 2.2: Landsat 8-9 Operational Land Imager (OLI) and Thermal Infrared Sensor (TIRS) bands [HY15].

2.1.2 World Glacier Inventory

The World Glacier Inventory (WGI) proves to be a useful resource for building our dataset, since it contains over 130,000 entries representing glaciers. Various parameters are stored in this file for each glacier, such as its geographic location in the form of latitude and longitude coordinates, total area, its elevation, orientation and many more. The dataset has been constructed through aerial satellite mapping,therefore the set can be viewed as a snapshot of the glacier distribution from 2012 [WGI].

There are a number of ways to retrieve data from the inventory:

- download the dataset in a single CSV file (wgi_feb2012.csv);
- search by parameter using the Search Inventory interface;
- extract regions through the Extract Selected Regions interface.

The CSV text file will be used with the purpose to define which are the glaciers to be included in the dataset to be built. An example of how this file looks like can be found in Figure 2.5.

	wgi_glacier_id	political_unit	continent_code	drainage_code	free_position_code	local_glacier_code	glacier_name	lat	lon
2	SU5X1430909	SU	5	X143	9	90	Zyuryuzamin	38.92	71.272
3	AT4J143OE00	AT	4	J143	OE	6	ZWISELBACH W	47.112	11.038
4	AT4J143OE00	AT	4	J143	OE	5	ZWISELBACH	47.11	11.052
5	CH4L01200008	CH	4	L012	0	8	ZWISCHBERGEN GL	46.108	8.041
6	CN5N236I0001	CN	5	N236	I0	1	Zuxuehui	31.828	94.675
7	CH4J14304001	CH	4	J143	4	1	ZUORT VADRET DA	46.738	10.271
8	CN5O282B002	CN	5	O282	B0	23	Zuogiupu	29.212	96.893
9	CN5O282A047	CN	5	O282	A0	476	Zuoguzasan	29.958	95.92
10	SU5X1430831	SU	5	X143	8	310	ZULUMART	39.13	72.78
11	CN5N224E001	CN	5	N224	E0	12	Zuima	29.839	96.456
12	SU5X1430948	SU	5	X143	9	489	Zotkin	38.649	71.244
13	SU5X1430949	SU	5	X143	9	490	Zotkin	38.649	71.244
14	SU5X1430932	SU	5	X143	9	326	Zordi-Birauso	38.673	71.664
15	NZ6B868B000	NZ	6	B868	B0	7	ZORA	-43.739	169.823
16	SU5T09106366	SU	5	T091	6	366	ZOPKHITO	42.88	43.43
17	IT4L01104020	IT	4	L011	4	20	ZOCCA S	46.285	9.647
18	IT4L01104021	IT	4	L011	4	21	ZOCCA E	46.292	9.653
19	AQ7SSI00012	AQ	7	SSI0	0	125	Znosko Glacier	-62.1005	-58.4865
20	SU4X0300190	SU	4	X030	1	903	ZNAMENITYY	80.53	61.02

Figure 2.5: World glacier inventory ASCII text file, as CSV

The parameters which will be extracted for the dataset construction are the following:

- **wgi_glacier_id**: unique id representing one glacier (or part of it, if the coverage area is larger);
- **glacier_name**: name of the glacier (if it has one);
- **lat**: latitude of the glacier;
- **lon**: longitude of the glacier.

2.1.3 Asset Acquisition

Since Landsat 8 acquires over 700 scenes per day, this means that there are over two million scenes available for download, either making use of already built user friendly tools or by simply querying for them directly.

USGS Earth Explorer

One of the most popular services for satellite imagery downloading is USGS Earth Explorer. This is used for querying and ordering of satellite images, aerial photographs, and cartographic products through the U.S. Geological Survey. The tool is particularly

useful when the main focus is to analyse a specific area rather than trying to acquire a large dataset of scenes. One can easily search for assets based on criteria such as world reference system path and row variables, latitude, longitude, cloud coverage, capture date and many others [USG].

However, downloading a large set of assets proves to be rather difficult by using this tool alone, since the parameters for each scene need to be manually set. On top of this, the query results have to be picked by hand and then passed for downloading through another application which handles their bulk download. This makes the process of building the dataset rather slow, frustrating and error prone. Such an example can be viewed in Figure 2.6, for the Belvedere glacier (45.942, 7.908).

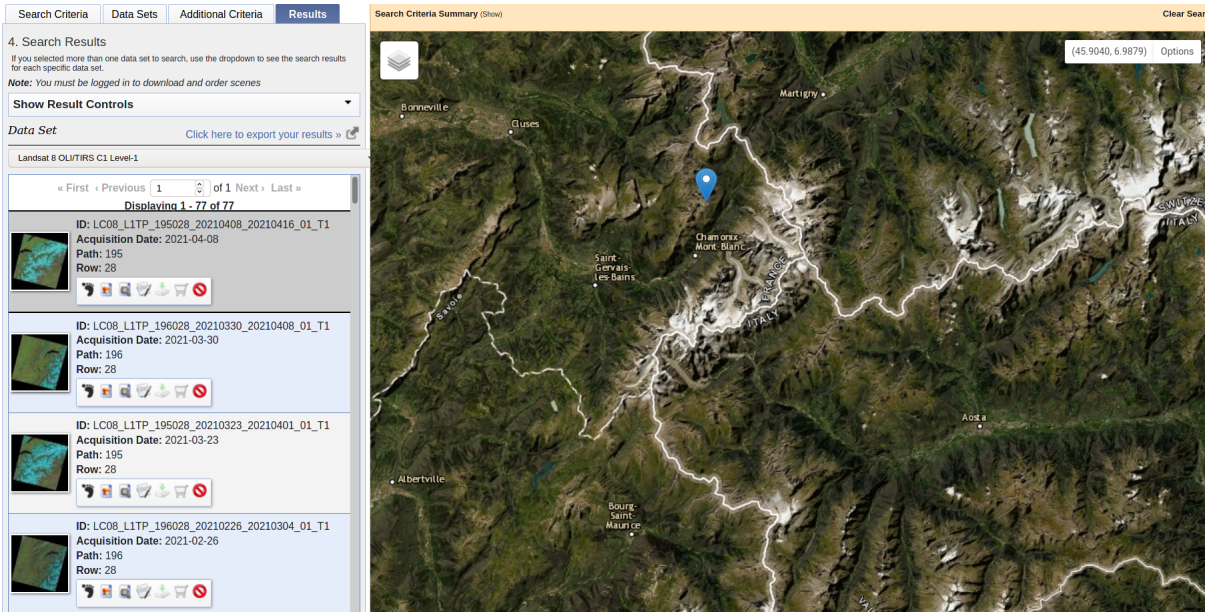

Figure 2.6: EarthExplorer

SpatioTemporal Asset Catalog API

In order to fix the problem of excessive manual labour which appeared by using the USGS Earth Explorer, we rather implemented an endpoint of the SpatioTemporal Asset Catalog API, specifically, the following: `http://nsidc.org/data/glacier_inventory/index.html` [STA]. The main idea of searching by using parameters still remains, but instead of manual inputting data for the search data, we rely on using the above-mentioned World Glacier Inventory ASCII text file, since it already has all the required information for each glacier.

By using this method we can pick which glaciers we want to download based on their coordinates and calculate a bounding box representing the area we want to search, required for the STAC API query. Since there might be clouds which could obfuscate the area of interest in the image, we also add a maximum allowed cloud coverage along the bounding box.

The STAC API query also requires a name for the collection of assets we want our queries to be made on, which for us is landsat-8-l1 (Landsat 8 Collection 1, Level 1). Using these three parameters we can now easily acquire a large number of assets with minimal manual labour, as compared to the more user friendly tool provided by USGS.

The downloaded assets will be stored at a user specified disk location and they will be structured as shown in the Figure 2.7.

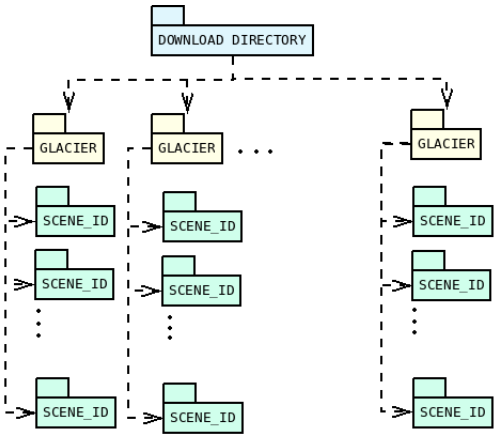

Figure 2.7: Download Directory

Landsat 8 Collection 1, Tier 1

To support analysis of the Landsat long-term data record, the Landsat archive was re-structured into a more formal data collection structure with the aim of keeping consistent information for each level 1 product. By making sure that these standards are met, the library can be used for various applications such as data stacking and time-series analysis [lc1]. By using data from this collection, we ensure that our images are fit for accurate pixel-to-pixel processing.

2.2 Dataset entities

We will further describe which are the bands necessary for the image generation and what are their uses. On top of this, we will further explain what is the normalized snow difference index and what is the optical flow used for.

2.2.1 Bands

Each Landsat 8 band is represented by a 16 bit grayscale image with a resolution between 7000 and 10000 pixels, each pixel representing 30 meters. We can conclude, therefore, that one scene covers around 200 and 300 km of Earth. Only the green and SWIR1 bands will be used for the purpose of this thesis and below we will discuss the specifications of each.

Band 3 - Green Band

Wavelength	0.53- 0.59 micrometers
Spacial resolution	30 meters
Resolution	between 7000x7000 pixels and 10000x10000 pixels
Depth	16-bit
Format	grayscale

Table 2.3: Landsat 8 green band specifications.

The green band, alongside with the red and blue ones, fall in the visible spectrum and it is usually used for mapping peak vegetation. Figure 2.8 is an example of the green band for the Belvedere glacier, specifically taken from scene LC81950282015098LGN01.

Band 6 - SWIR1 Band

The shortwave infrared 1 band is particularly useful for enhancing object which look similar in other bands, such as soils and rocks [bd]. Alongside this, it also discriminates moisture content of soil and vegetation and penetrates thin clouds [HY15]. Figure 2.9 is illustrated below as an example of a SWIR1 band taken from the same scene as the green one above.

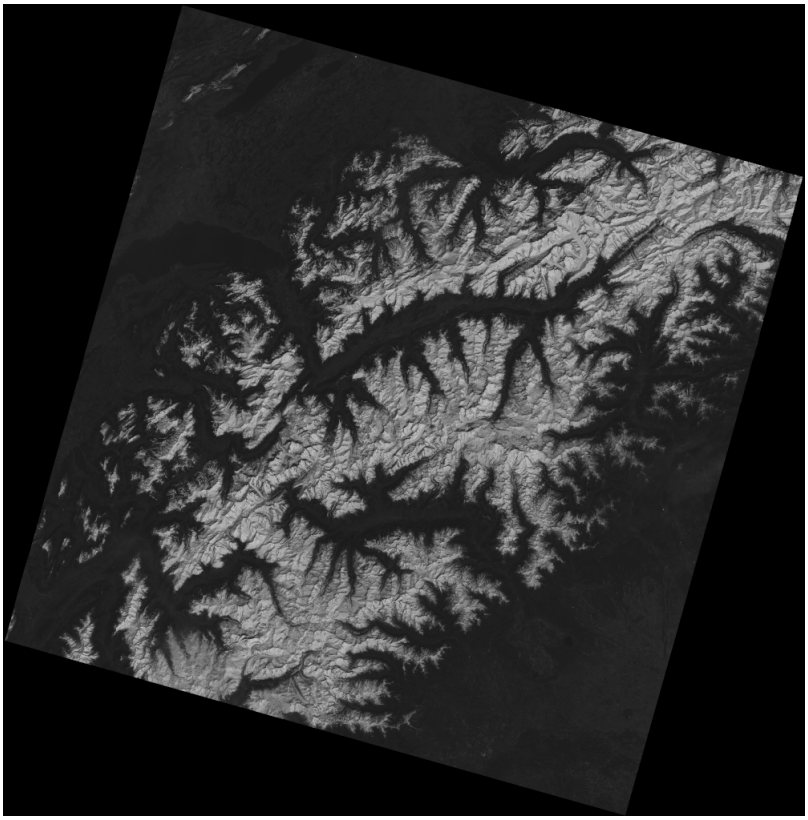

Figure 2.8: Green band of scene LC81950282015098LGN01 from the Belvedere glacier, Italy.

Normalized Snow Difference Index

The normalized snow difference index (NDSI) is an index which relates to the presence of snow/ice in a pixel. Snow and ice usually have a very low reflectance in the shortwave infrared spectrum and very high in the visible one, which is useful for mapping out most types of clouds from the scene [nds]. We can therefore use the formula from Equation 2.1 in order to highlight the snow and ice pixels from a Landsat 8 image.

$$NDSI = \frac{green - SWIR1}{green + SWIR1} \tag{2.1}$$

By combining the snow reflectance behaviour of snow and ice for each of these bands, we can create a normalized snow difference index image which has values in the range [-1, 1]. We can then apply a threshold on the pixels which states that if the NDSI value for a pixel is larger than 0.0, then that pixel represents snow/ice covered land; similarly, if its value is smaller than 0.0, that pixel represents snow/ice free land [nds], [Hal15], as represented in Equation 2.2. Of course, this represents the general value for the threshold,

Wavelength	1.57 - 1.65 micrometers
Spacial resolution	30 meters
Resolution	between 7000x7000 pixels and 10000x10000 pixels
Depth	16-bit
Format	grayscale

Table 2.4: Landsat 8 SWIR1 band specifications.

but with the increase of its value, we can more accurately differentiate between snow and ice. With larger threshold values we can state that a pixel represents ice covered land rather than just snow fall land [Hal15]. We have tried different values for the threshold and came to the conclusion that a value of 0.3 is best suited for our needs.

$$\begin{cases} \text{snow/ice land} & \text{if } NDSI \geq 0.0 \\ \text{snow-free land,} & \text{NDSI ¡ 0.0} \end{cases} \tag{2.2}$$

An example of a generated NDSI for the Belvedere glacier, scene LC81940282013341LGN01, can be viewed in Figure 2.10.

The image is colour coded in order to enhance the difference between pixels which are considered either snow or ice (white), and the rest of the terrain (green).

2.3 Alignment

Landsat's trajectory orbiting Earth is not fully precise, therefore not all images will be pixel-to-pixel aligned, which is a problem for image processing. Since we want to track each pixel's movement, we must be sure that its coordinates in the image do not change between any two given scenes. Figure 2.11 highlights the misalignment from scene between scenes LC81950282013316 and LC81950282013364, as an example.

Since the bands we work with have a spacial resolution of 30 meters, which means that each pixel in the image represents 30 meters. Even with a misalignment of just 50 pixels we would end up with a 1.5 km difference between two scenes. Tracking pixels through the image without aligning them first would mean that we could never be sure that what we are looking at is indeed the same location. We have used different approaches on alignment which will be described in Section 3.2.1. Figure 2.14 highlights the alignment

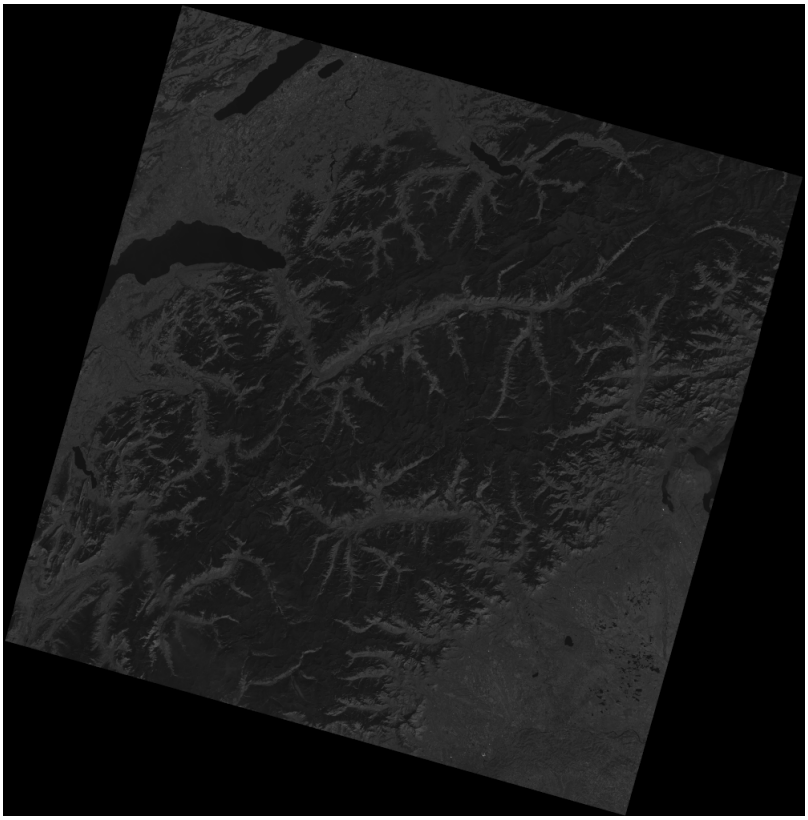

Figure 2.9: SWIR1 band of scene LC81950282015098LGN01 from the Belvedere glacier, Italy.

corrected scene from Figure 2.11.

NDSI's Optical Flow

Since our dataset represents a **time series of satellite images**, as described in Section 2.1.1, we thought that in order to generate a new image of this series, we could extract the **motion of each pixel** from one scene to another. This information could then be applied between any two consecutive (date-wise) scenes from the set and we could update a pixel's coordinates based on the value its motion vector and store it in a new image.

Extracting the motion vectors between two consecutive frames can be achieved by calculating their optical flow. **Optical flow** is defined as the motion of objects between consecutive frames of sequence, produced by the relative movement between the object and camera. By using computer vision algorithms which calculate the optical flow of two scenes, we could track the motion of melting glaciers across them in order to estimate

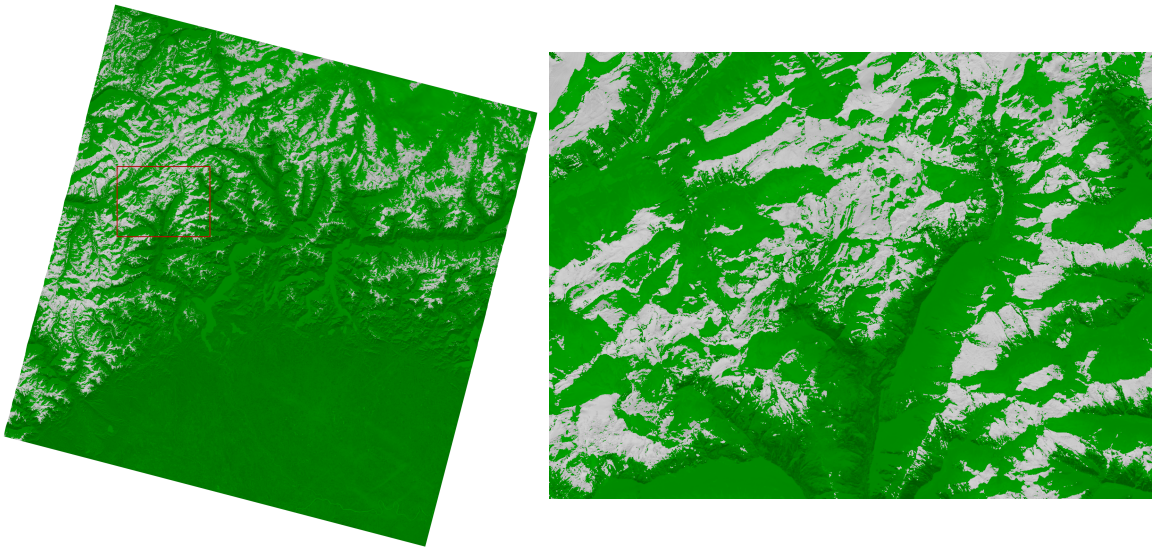

Figure 2.10: NDSI image of scene LC81940282013341LGN01.

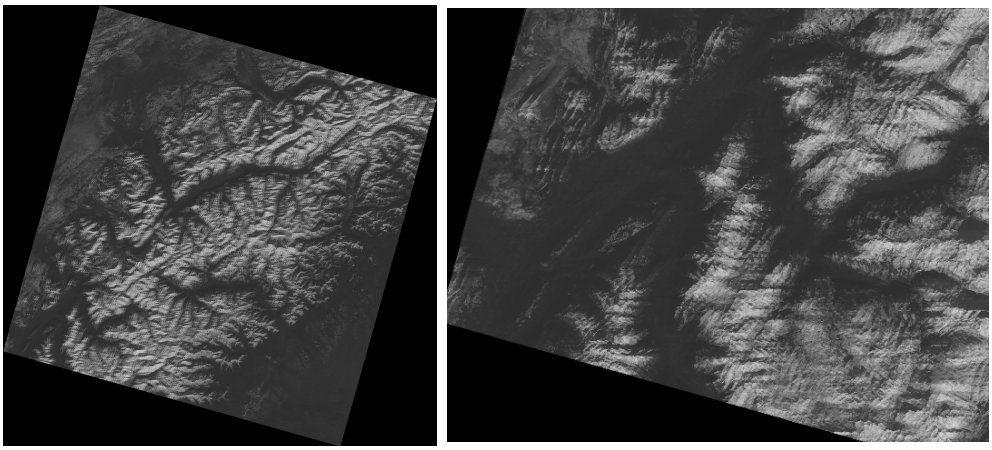

Figure 2.11: Overlapped unaligned green band of scene LC81950282013316 and reference LC81950282013364 of the Belvedere glacier (195/028 WRS-2).

their current velocity and possibly **predict their position** in the next frames [opt].

Figure 2.13 emphasizes the problem visually, where we can express an image as a **function of space**, with the coordinates (x, y), and **time** t. If we take the first image $I(x, y, t)$ and we move its pixels by a distance of dx, dy over a timestamp dt, we obtain the new image as follows: $I(x + dx, y + dy, t + dt)$ [orb].

There are multiple types of optical flow algorithms, but for the purpose of our thesis, we have chosen the **dense optical flow** one, specifically **Gunnar Farneback's**. Even if dense implementations have higher cost we chose to make this trade mainly because it calculates the motion for each pixel of the frame and it also has a higher accuracy [orb] compared to methods such as Lucas-Kanade (sparse) [LK81].

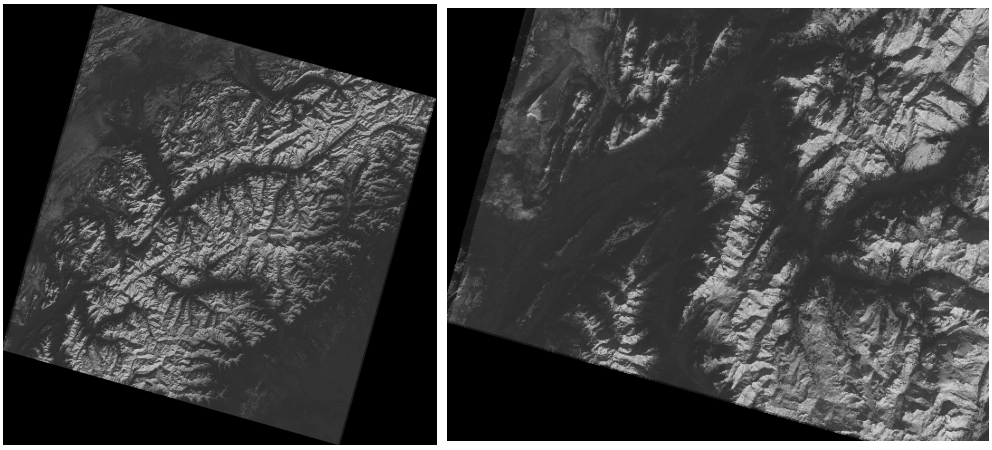

Figure 2.12: Overlapped aligned green band of scene LC81950282013316 and reference LC81950282013364 of the Belvedere glacier (195/028 WRS-2).

We have used optical flow as a tool to generate **distance vectors** based on motion between the NDSI($time = t$) and NDSI($time = t + dt$). These vectors will be used in order to create the **motion predicted NDSI**.

Motion Predicted NDSI

Whilst optical flow allows us to see movement of the ice front which already happened in the past, we wanted to use this information and apply it on the most recent images such that we can estimate further glacier movements.

We obtain the motion predicted generated NDSI by relocating each pixel value from the NDSI($time = t + dt$) to a **predicted location**. For now, this location is generated by adding the distance vectors obtained by optical flow as described in Section **??** to the pixels of the same image, therefore shifting them by twice as much as they originally did, in their respective directions, as it can be observed in Figure 2.15. More methods of calculating the predicted location can be found in Section 5.1.

By applying this function to each pixel of the NDSI ($t + dt$) image and by making some adjustments which are further described in Section 3.2.4, for the Belvedere glacier, scene LC81940282015363LGN02, we have generated the motion predicted NDSI as shown in Figure **??**. The ice coverage for the generated image is 5.0991% while the one for the actual NDSI($time = t + dt$) is 5.3066%.

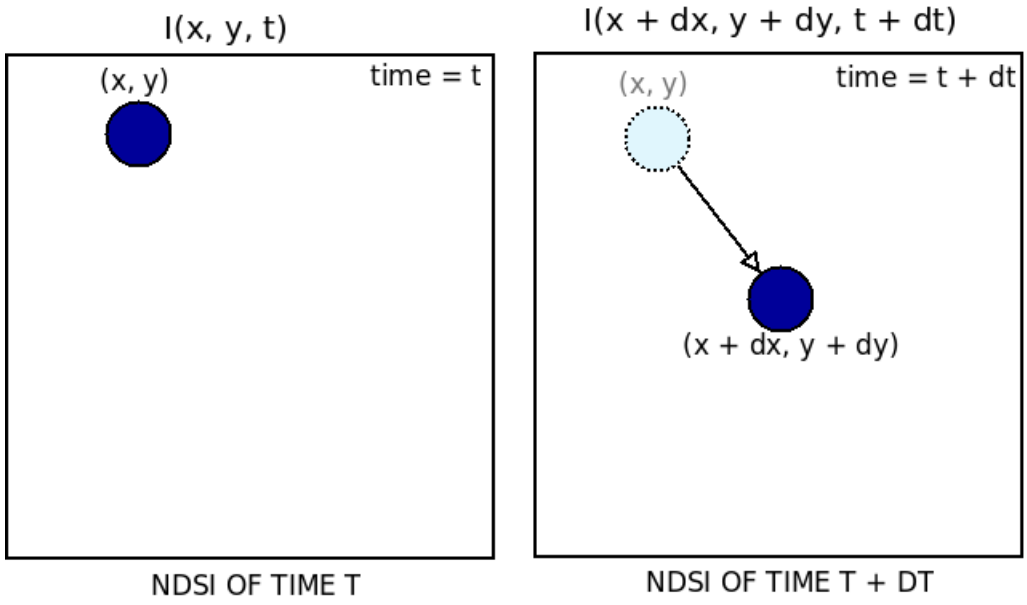

Figure 2.13: Optical flow problem visualisation.

2.4 Graphical User Interface

2.4.1 Search and Download

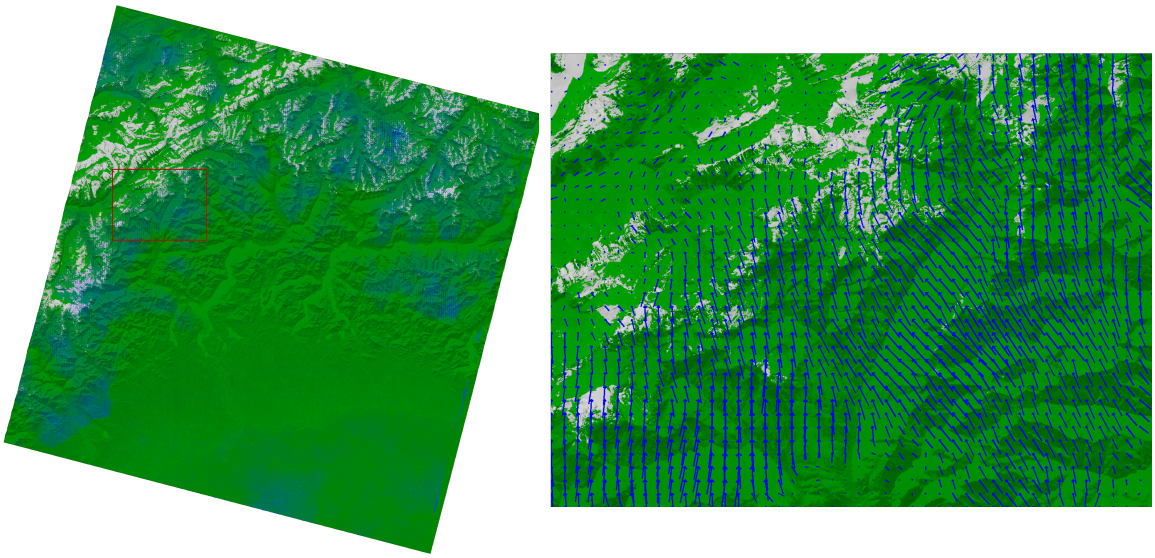

Figure 2.14: Optical flow vectors, where the previous image is LC81940282013341LGN01 and the current image is LC81940282013341LGN01.

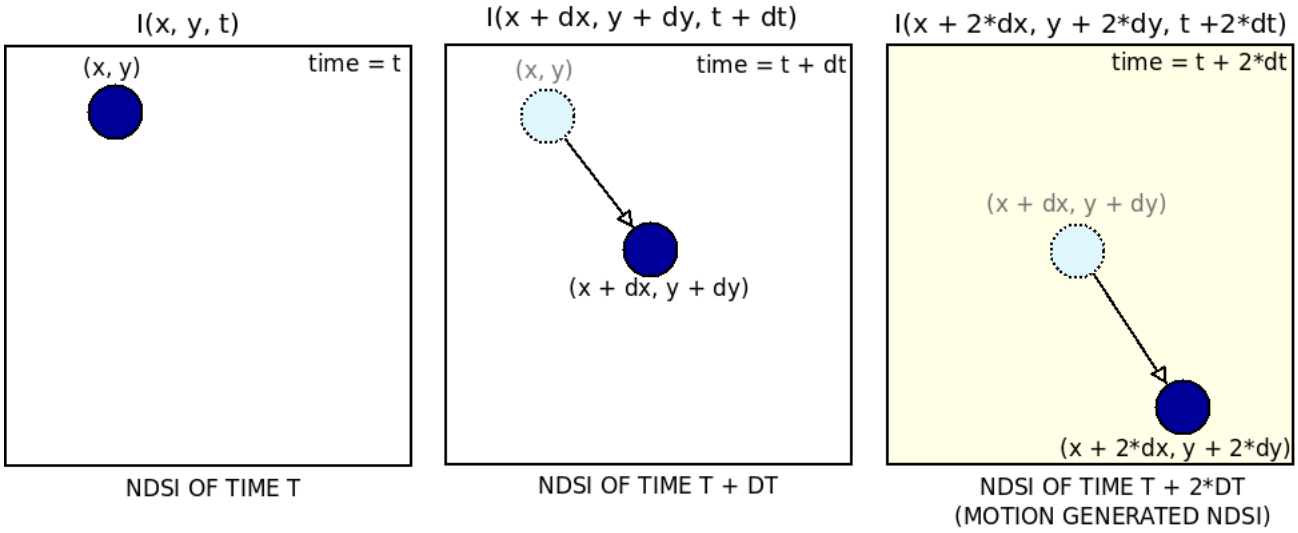

Figure 2.15: Motion generated NDSI algorithm.

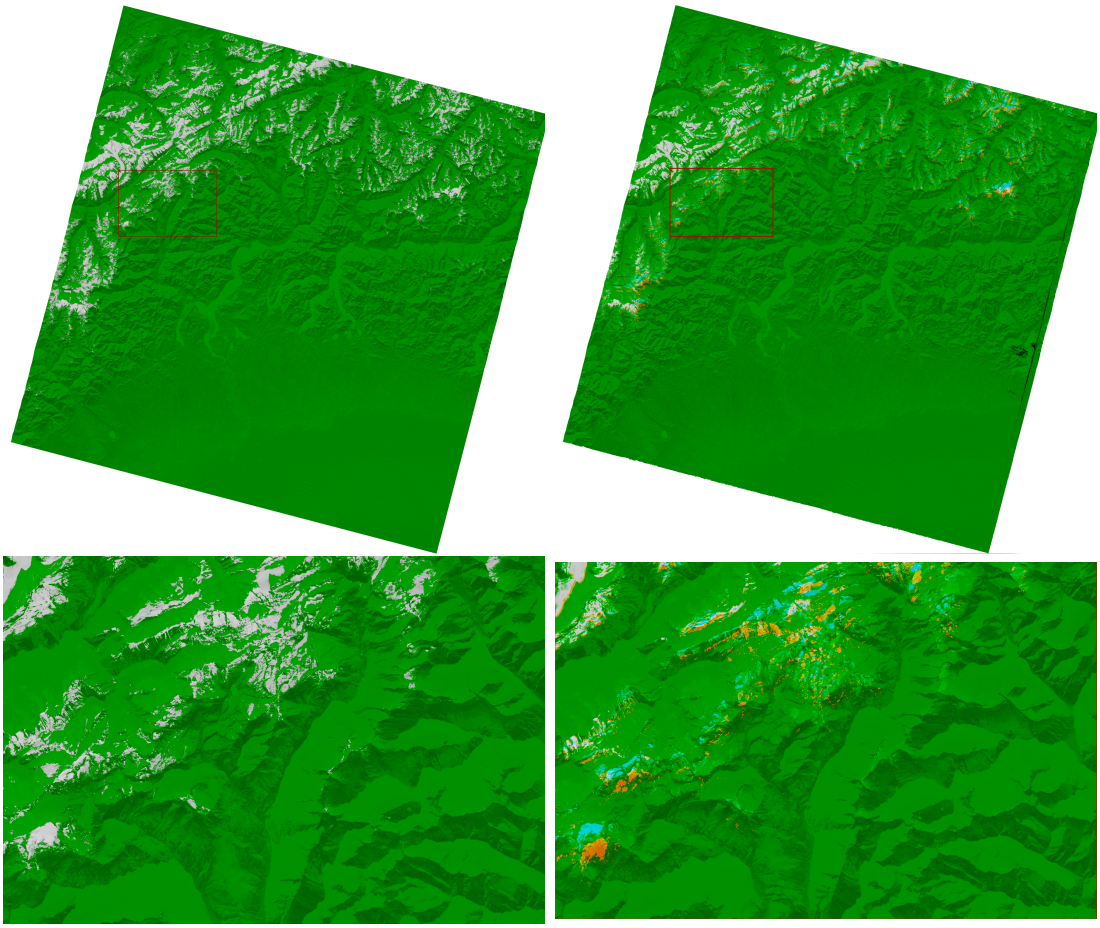

Figure 2.16:

Chapter 3

Design and Implementation

The design of the application will be discussed while focusing on two different aspects:

- **search and download**
- **processing**

The first section is optional and can be run independently from the processing, since the user can have an already created database of images. However, we have focused on simplifying the process of satellite imagery downloading as described in 2.1.3 through using the sat-search library as a helper for searching and downloading assets. More information on querying can be found in 3.1. Given an existing set of data, the next step is to pass the root folder which contains the glacier directories to the processing unit, also designed as a plug-in mechanism which will trigger the graphical user interface to pop up and allow for the processing options to appear. More information on this section can be found at 3.2. From there, one can start processing by simply making use of the predefined buttons described in 2.4.

3.1 Search and Download

Searching as well as downloading have been implemented on top of the sat-search library and it is used directly from the command line interface by running the script described in Section 2.4.1.

As an input we will be using a CSV file which will have the form as described in Section 2.1.2. Mainly we will need just four attributes to be specified for each desired

glacier in order to create a query for searching, as follows: *wgi_glacier_id, glacier_name, lat and lon.*

The CSV file will be intercepted by the **Download** class and sent for processing row by row (glacier by glacier) to the **GlacierFactory** class. This one is responsible for parsing each row of the input CSV file and transforming the information in **Glacier** objects, which will be passed back to the **Download** class.

Using the newly created Glacier object we can construct its **search query** by calculating its bounding box, specifying the asset collection from which we request data and setting other parameters (maximum allowed cloud coverage, in our case). Listing 3.1 represents an example of querying for glacier Belvedere, with the following parameters set in the CSV file: *"IT4L01211009","BELVEDERE","45.942","7.908"*.

Listing 3.1: Search query created by sat-search

```
{"page": 1, "limit": 170, "bbox": [7.907990000000001, 45.94199, 7.90801,
    45.94201], "query": {"eo:cloud_cover": {"lt": 10}}, "collection":
    "landsat-8-l1"}
```

The result of each glacier query will be automatically saved in a **JSON file** which is used as a data buffer between the search and download. The downloader takes each JSON file generated by the search engine and sends the command for getting that asset.

The technical specifications of the Download, GlacierFactory and Glacier classes can be found in Figure 3.1.

Data corruption verification Both the searching and downloading functions are executed concurrently, since they are **time intensive** tasks. Each band has around 60 MB, which would make one scene approximately 360 MB. For the Belvedere glaciers, we found 78 scenes with a cloud coverage of 10%. This means that the size of the entire glacier data will be around 28 GB. Given that one asset's size is quite large, it takes a lot of time to finish the download. In this time, even a small connection interruption might result in corrupted data files, which would interfere with processing. Therefore, we implemented an **extra security measure** in the sat-stac library which verifies whether a file with the same name already exists at the download location. If it does, its size will be compared to the size taken from the STAC-API servers. In case the two file sizes don't match, it means that the file got corrupted during download and it will be downloaded

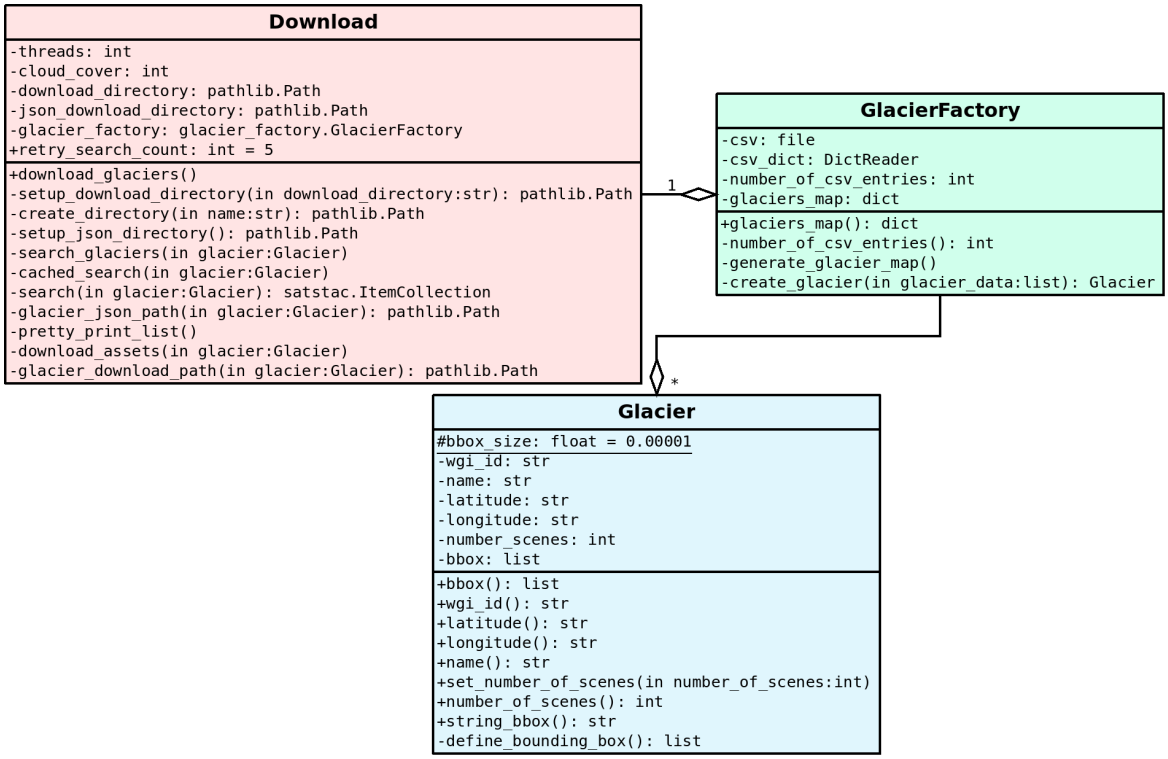

Figure 3.1: Technical specifications of Download, GlacierFactory and Glacier classes.

again. If the sizes match, the downloader will skip the file and continue when it finds one requested file which was not yet found on the disk, such that we do not end up unnecessarily overwriting the already existing files in case of failure.

3.2 Processing

Processing can be viewed as four different entities, as following:

- Image alignment;
- Normalized snow difference index;
- NDSI's motion matrix;
- Motion generated NDSI.

Before generating any images, we first need to make sure that the pixels between any two scenes are not misaligned. The details of the alignment process are described in Section 3.2.1. After we ensure that out images are fit, we move on to creating the

normalized snow difference index image in order to enhance the ice through a set threshold (see Section 3.2.2). We then create the matrix of pixel motions calculated between two consecutive (date-wise) scenes, process described in Section **??**. Based on the information of where each pixel has moved between two consecutive images, we can then create our future NDSI image by applying the motion vectors on each pixel from a scene. More details on the design and implementation of this part can be found in Section 3.2.4. The normalized snow difference index image will be computed for each scene as described in Section 2.2.1.

3.2.1 Alignment

In order to keep a high level of abstraction, we have split each scene into aligned and unaligned: Scene and AlignedScene objects. These and their children are created when crawling through the glaciers directory, specifically for each region of interest. However, aligning all the images when crawling would result into a lot of idle time for the user when starting the application and it would not be overall feasible to do so. Therefore, a scene is aligned only when it is specifically being selected in the graphical user interface (or when another entity needs it, such as optical flow and image generation). By doing this, we ensure that there is no unnecessary extra waiting time for each scene that we have when powering up the interface.

As for the actual alignment, we have used an algorithm which **collects features** from each band of a scene and tries to match them with a given reference one's features. The obtained **matches** can be then used to create an **affine transformation matrix** which specifies by how much did the current image **rotated and translated** in comparison to its reference. The affine transformation matrix will be then applied to the current image by a **warping** algorithm with the goal of ensuring as much as possible that there is no misalignment between any pixel of two different images from the same time series. Section 3.2.1 describes more technically how we built this and what were the computer vision algorithms used for that matter. Figure 3.2 contains the technical details of each class which takes part into the alignment process.

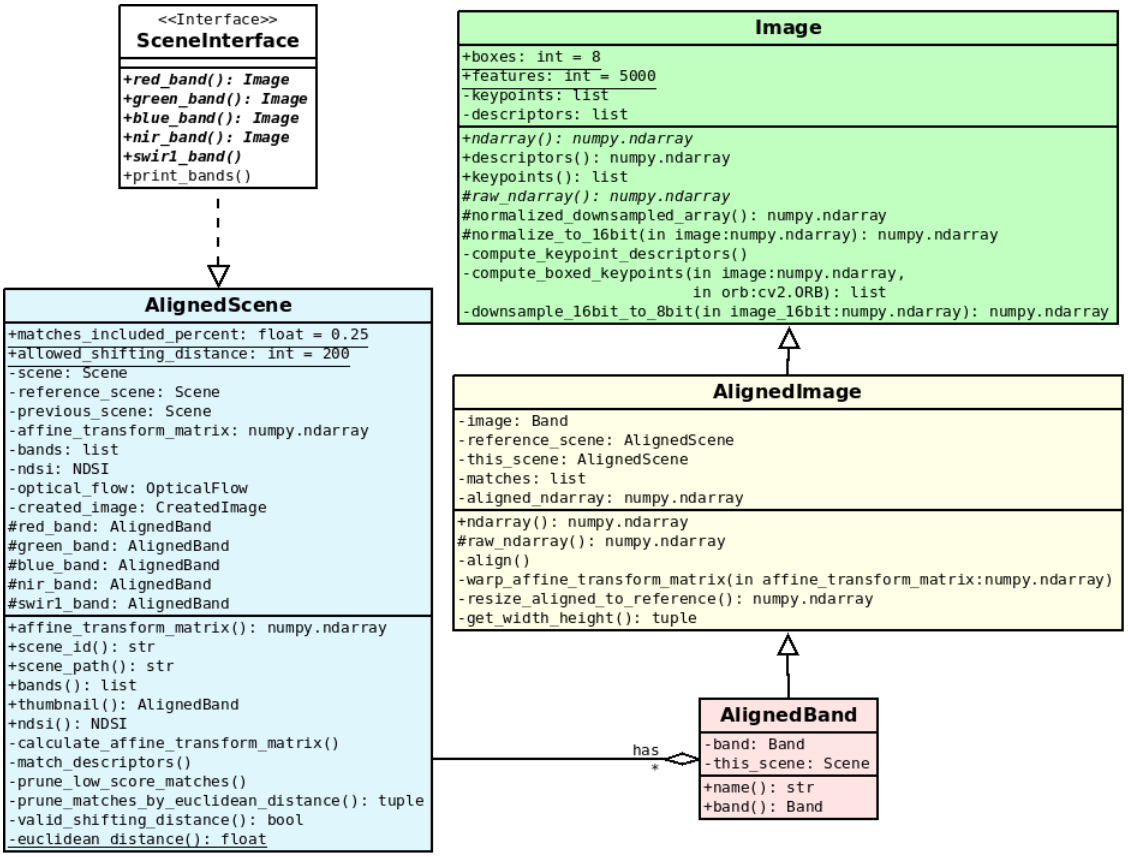

Figure 3.2: Technical diagram for the SceneInterface, AlignedScene, AlignedBand and Image classes.

Alignment algorithms

As we have seen in Section 2.1.1, the Landsat 8 satellite does not have a perfect trajectory, which results into misaligned images. These changes are not obvious to the naked eye from scene to scene, but overlapping two scenes highlights this problem, as we can see in Figure 2.11. Even so the scenes are almost similar, which means that aligning them does not prove to be very hard and it is done by using a strong **keypoint detection algorithm** which in our case detects edges formed by mountains and other geographical features present in the scenes. The **computer vision algorithms** used for this are **ORB (Oriented FAST and Rotated BRIEF)**, **Harris Corner Detector** and **RANSAC (Random Sample Consensus)** and they are applied on the raw 16bit grayscale image.

ORB represents a fusion between the features from accelerated segment test (**FAST**) keypoint detector and binary robust independent elementary features (**BRIEF**) descriptor. The FAST detector finds keypoints in the image and uses Harris corner measure to

select the a number of top points from the list (25%, in our case) [orb]. In order to use the detector we must first normalize and downsample the 16bit raw image to 8 bit, as it is required by ORB. Each keypoint is then represented by a circle of 16 pixels. The descriptors must then identify these keypoints and pair with each other. We compute for each scene its **keypoints and descriptors** taken from **each band**, in order to increase the precision of alignment later on. Also, since we are working with satellite imagery there might be cases when the algorithm only finds keypoints in a small part of the image, resulting in erroneous distortion. **Splitting** the image into multiple boxes and applying the orb detect algorithm on each separate part of the image proved to solve this problem and create much better results.

After ensuring that we have good enough keypoints we will be aligning each scene with its reference by simply comparing the keypoint-descriptor pairs between the two scenes and checking which are equal. If two pairs are found to be equal, it means that the algorithm has found the same feature in both images and can calculate by how much it **rotated and shifted**. However, not all of these matches represent good results in our case, since the trajectory Landsat 8 can vary. We have found that selecting only **5000 keypoints** in the **top 25% matches** from the total yielded in the better results overall. We then set the **maximum allowed shifting euclidean distance** between any two pixels to be at most 200, so 6km on the map, therefore getting rid of outlier matches based on the distance.

The **matches** from the reference and image to be aligned will be used alongside the **Random Sample Consensus (RANSAC)** algorithm in order to create the **affine transformation matrix** using the cv2 library. RANSAC, proposed by Fischler and Bolles [FB81], is a general parameter estimation approach designed to handle a large proportion of outliers in the input data [Der10]. After we have the transformation matrix, we can **warp** it on the image to be aligned in order to get the result.

3.2.2 Normalized Snow Difference Index

The NDSI is generated by using the Formula 2.1 described in section 2.2.1 on the green and shortwave infrared bands from a specific scene. Each band pixel is converted to **float32** in order to increase the **precision** of calculation; as each band is read as a n-dimensional array we can use numpy's optimised processing for generating the NDSI index image, since it converts the data and runs on native code rather than going through

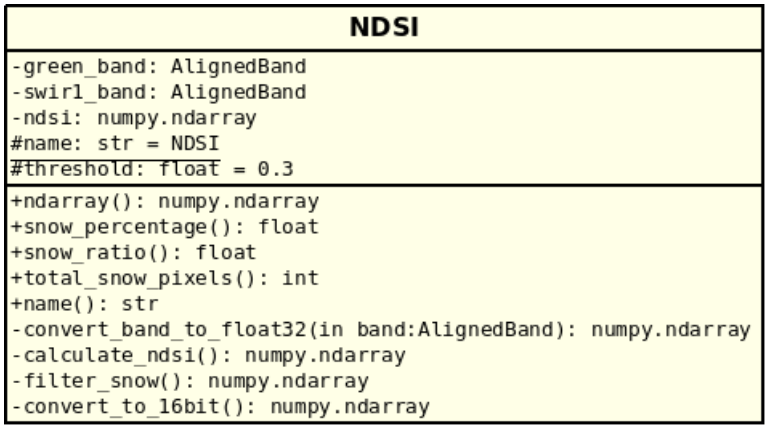

Figure 3.3: Technical diagram for the NDSI class.

Python's interpretor. We then **filter** the generated image such that everything which is snow-free land will be -1 (**black**), but this is only applied for better visual analysis. In the background, we work with the full range of the image for better precision, which is the raw image.

Figure 3.3 holds the technical diagram for the NDSI.

3.2.3 NDSI's Optical Flow

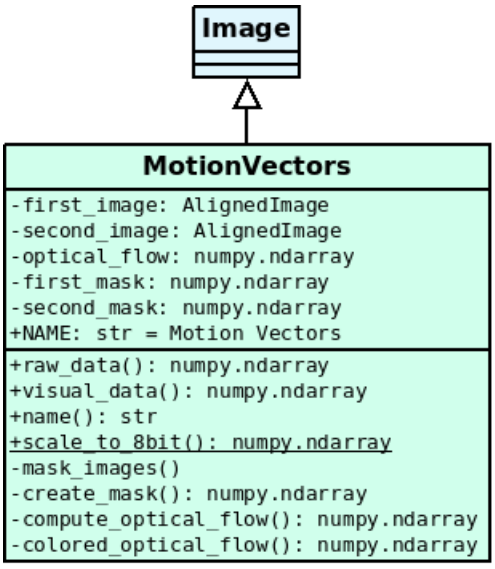

Figure 3.4: Technical diagram for the MotionVectors class.

As described in Section 3.2.4, in order to determine the motion vector for each pixel between two images, we have used **Gunnar Farneback's** algorithm for dense optical

flow calculation, implemented by the **OpenCV** library. The MotionVectors class takes as input two NDSI images.

Since the NDSI images have been aligned, depending on the affine transformation matrix, they will not overlap perfectly any more, resulting in **artifacts at their borders**. Applying optical flow by using two NDSI images as such would result in highly distorted motion vectors at the borders. For fixing this, we have created a **mask** from each image such that both of them are cropped with the mask of the other. This results in losing a small number of pixels at the borders which could not be taken into consideration anyway.

In addition to the two NDSI image inputs, we have used the **optical flow algorithm** with the **parameters** specified in Listing 3.2. We have used a **pyramid scale of 0.5** and **6 pyramid levels** having in mind that the images that we work with are large. By choosing this combination of parameters, we state that at each level, the image is going to be reshaped at half the size of the previous one; therefore the area of search for motion is small enough such that the optical flow is able to track movement.

Listing 3.2: Optical flow parameters

```
cv2.calcOpticalFlowFarneback(..., pyr_scale=0.5, levels=6, winsize=15,
    iterations=3, poly_n=5, poly_sigma=1.2, flags=0)
```

After our images are perfectly overlaid, we can give them as inputs to the optical flow algorithm. The result of this will be the computed **motions** for each pixel, represented as a tuple of the **distance** that its coordinates moved from one frame to another. Specifically, as referring to Figure 2.15, each item from the optical flow n-dimensional array will represent the **motion distance vector** (dx, dy) as calculated between **NDSI**$(time = t)$ **and NDSI**$(time = t + dt)$.

To be able to visually interpret the optical flow output, we have used a **colour coded image**. Colour intensity is directly proportional to the motion vector length, while its hue represents the direction of this vector, as represented in Figure 3.5.

3.2.4 Motion Predicted NDSI

As means to put in practice what we described in Section 2.3, we start by initialising a new numpy n-dimensional array based of the **shape** of the image **NDSI**$(time = t + dt)$. The new array will be filled with the data generated by using the two required entities:

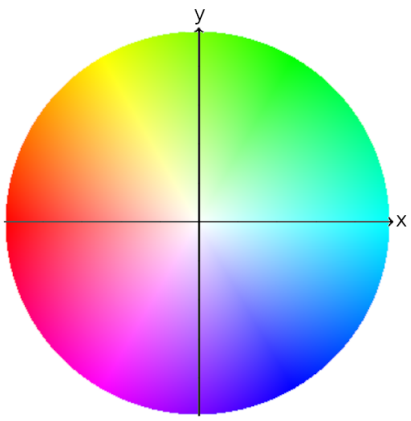

Figure 3.5: Hue representation of the optical flow.

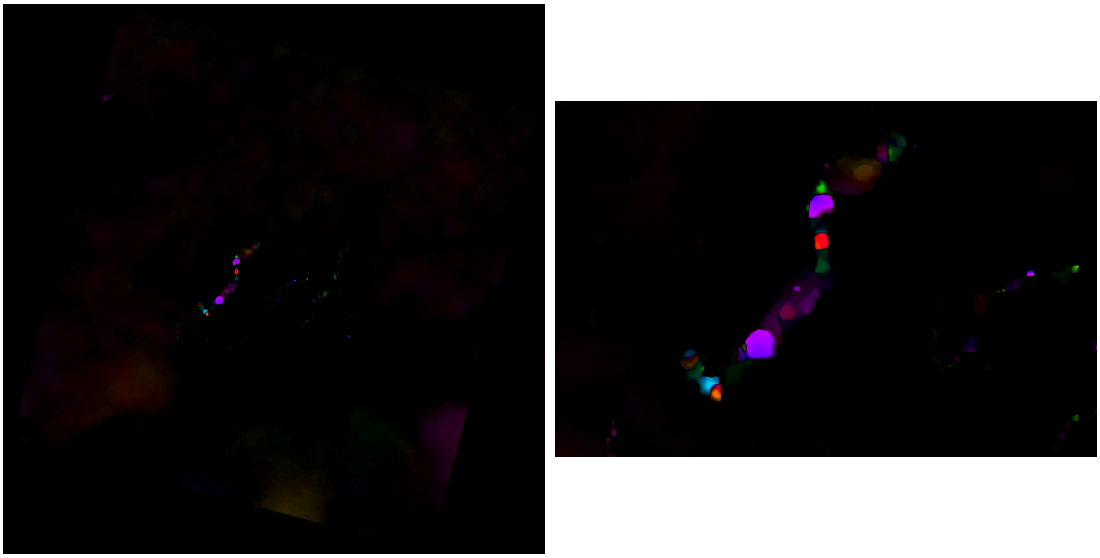

Figure 3.6:

- **NDSI**($time = t + dt$);
- **motion vectors** (dx, dy) extracted by optical flow.

Each **motion vector** (dx, dy) corresponds to the movement of a pixel from the NDSI($time = t + dt$) image. We can use this information to **generate** the new **predicted position** $(x + 2 * dx, y + 2 * dy)$ for each pixel of the NDSI($time = t + dt$) image.

As a first approach of populating the new image, we have simply **iterated** over all the pixels in the NDSI($time = t + dt$) image and calculated their **new coordinates** based on their **motion vector** information, as it can be seen in Algorithm 1. Since this is an iterative approach, it does not scale for images as large as the ones we are using.

Algorithm 1: Algorithm used for motion predicted image generation based on the optical flow vectors and NDSI($time = t + dt$)

```
function generate (previous_NDSI, motion_vectors);
  Input  : The NDSI(time = t + dt and the motion vectors generated by optical
           flow between NDSI(time = t) and NDSI(time = t + dt)
  Output: The motion predicted NDSI(time = t + 2 * dt))
motion_predicted_image ← previous_NDSI.shape;
width ← NDSI.width();
height ← NDSI.height();
for y in [height, width] do
    for x in [height, width] do
        dx ← motion_vectors[y][x][0];
        dy ← motion_vectors[y][x][1];
        new_x ← x + dx;
        new_y ← y + dy;
        motion_predicted_image[new_y][new_x] ← previous_NDSI[y][x];
    end
end
return motion_predicted_image;
```

On top of this, since we are using the Python language, which is an interpreted one, each operation has to go through an interpreter before being run. For very granular data processing this proves to be **inefficient**. For a typical image of 8543x8039 resolution, it takes as much as 10 minutes for generating the image on the machine that I have used (specifications at Section 4).

Since using the naive approach is not feasible in our application, we had to avoid iterating over the whole image in the first place. Therefore, instead of looping over each pixel in order to generate the values for the new image, we have made use of the **numpy** library to **heavily optimise** our data processing. By using numpy functions and vectorized operations directly instead of iterating we were able to improve the processing time by around 1700%, bringing it down approximately from 10 minutes to 35 seconds.

In the iterative approach, for each pixel we add its motion to its position such that we get its new location. These numbers can be computed ahead of time and transformed into arrays such that we only use numpy operations. By creating an **array of the initial coordinates** x, y and **adding the motion vectors** (dx, dy) array to it, we **generated the absolute coordinates** where each pixel from the NDSI image should be translated. By adding this modification we got rid of lines 5, 6, 7, 8, 9, 10 and 11 from the algorithm and replaced them with the code as it can be seen in Algorithm 2. The $absolute\_coordinates$ and $previous\_NDSI$ can be treated as a sparse array which is then densified using numpy capabilities.

The technical description of the MotionPredictedNDSI class can be found in Figure 3.7.

The function used to generate the predicted NDSI does not have a **one-to-one domain**, thus making it undefined for certain inputs. This results in a very **noisy** generated image, as it can be seen in Figure Figure 3.8. We have implemented a **filter** which uses the weighted average of the neighbouring pixels values to fill the missing ones. The detailed description of the implementation can be found in Section 3.2.5.

3.2.5 Generated image filtering

A first step into creating the filter for the motion predicted NDSI image is to create a **mask** which will be applied on the black border of the scene such that we are looking for **undefined values** only inside it. Using numpy, the **coordinates** of these pixels are extracted into an array which then can be processed in parallel by Pythons' **multipro-**

Algorithm 2: Improved algorithm used for motion predicted image generation based on the optical flow vectors and NDSI($time = t + dt$)

function generate ($previous\_NDSI, motion\_vectors$);

Input : The NDSI($time = t + dt$ and the motion vectors generated by optical flow between NDSI($time = t$) and NDSI($time = t + dt$)

Output: The motion predicted NDSI($time = t + 2 * dt$))

$motion\_predicted\_image \leftarrow previous\_NDSI.shape$;

$width \leftarrow NDSI.width()$;

$height \leftarrow NDSI.height()$;

$index\_array \leftarrow [[[0,0],[1,0]...[width,0]\ [0,1],[1,1]...[width,1]\ ...\ [0,height],[1,height]...[width,height]]]$;

$absolute\_coordinates \leftarrow motion\_vectors + index\_array$;

$motion\_predicted\_image[absolute\_coordinates] \leftarrow previous\_NDSI$;

return $motion\_predicted\_image$;

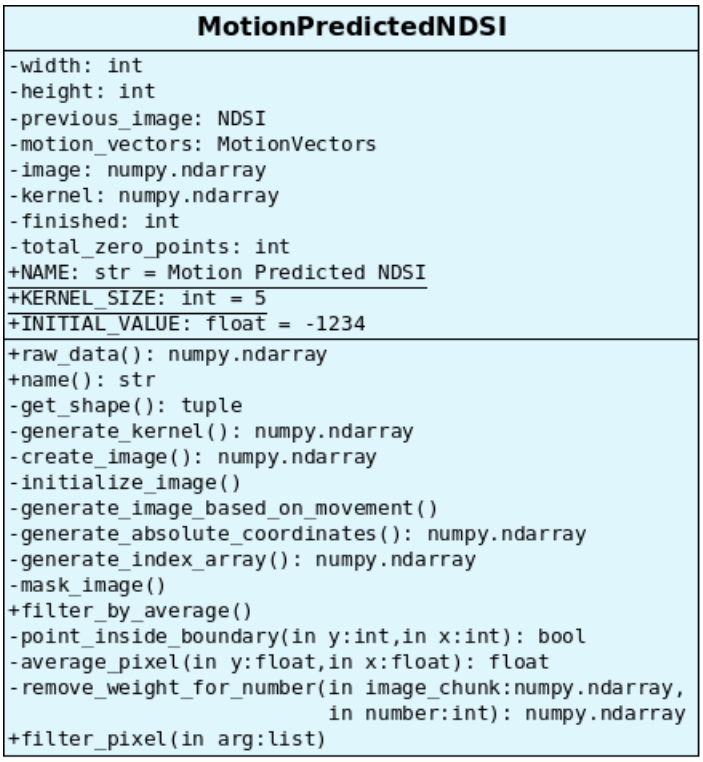

Figure 3.7: Technical diagram for the MotionPredictedNDSI class.

cessing library. We have used multiprocessing instead of threading because Python does

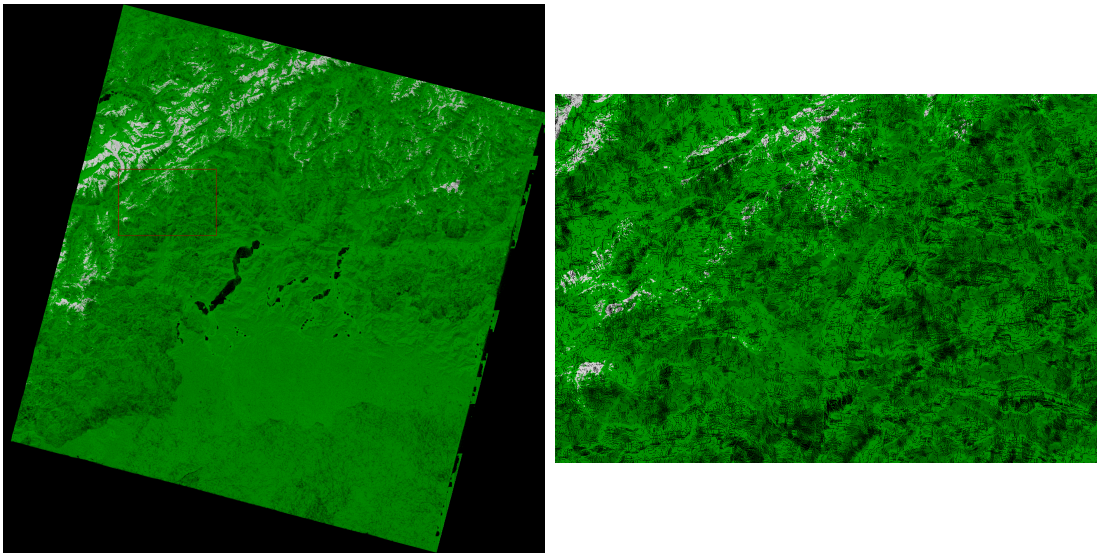

Figure 3.8:

not have true parallelization in multithreading, only concurrency, which would not be a real improvement on the processing time. Since all the processes have to write different chunks of the same image and they do not share the same memory space, we chose to solve this problem by creating a shared memory buffer to hold the image.

The value of each found undefined pixel has to be calculated as an **weighted average** composed of its **neighbouring pixels**, as shown in Figure 3.9 top. We created the **kernel** which holds the weight of each pixel in the neighbourhood such that pixels closer to the centre have a higher weights than those near the edge. However, we have the case when we have multiple undefined pixels in the same neighbourhood. Since we cannot take their values in consideration, we do not want to include them in the weighted average, thus we set their weight to be 0, as shown in Figure 3.9 left, bottom. The result of the weighted average will be then stored as the value of the currently focused undefined pixel, as highlighted in Figure 3.9 right, bottom.

3.3 Extras

Below we will talk about the

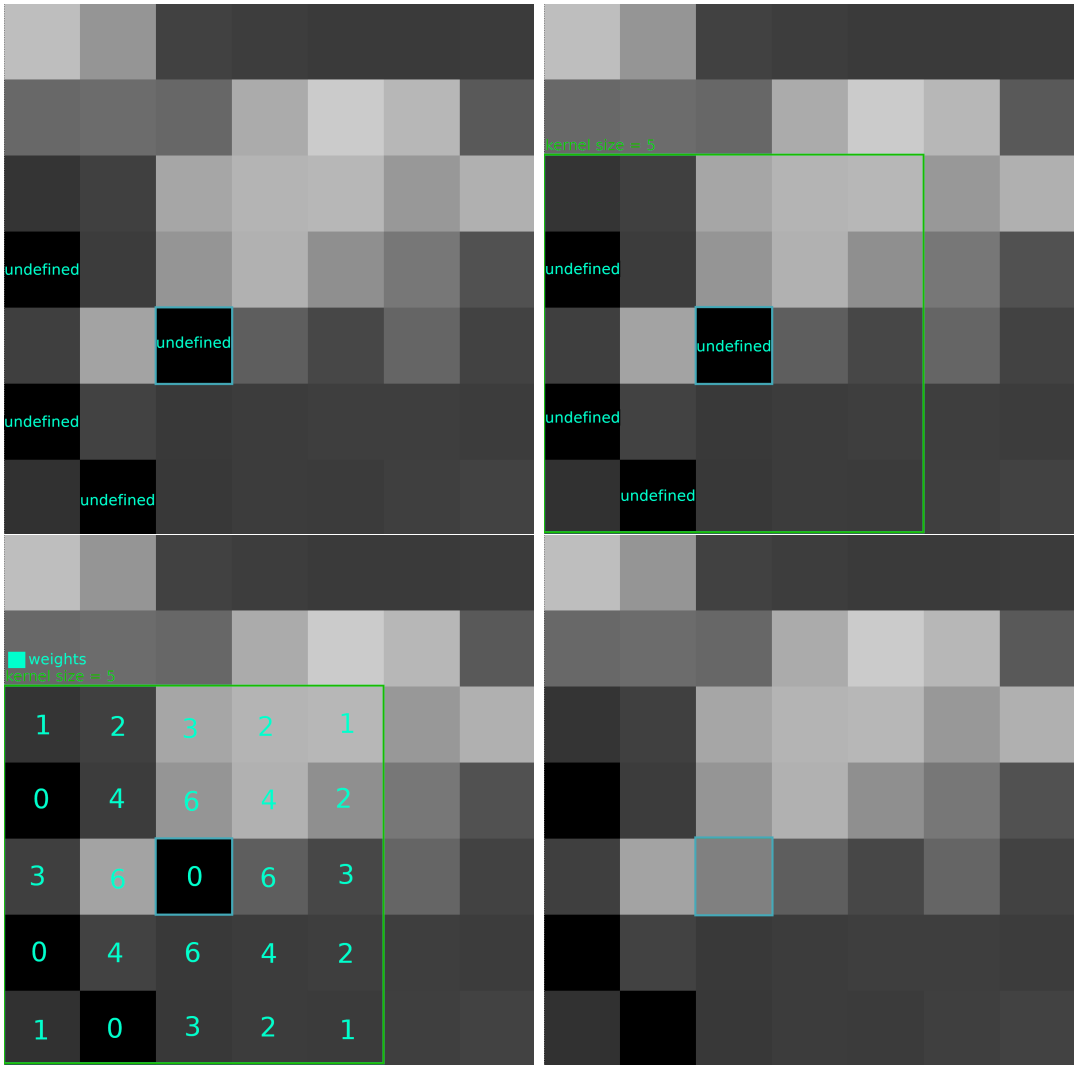

Figure 3.9: Zoomed in part of the predicted NDSI (left, top). Neighbourhood extracted (right, top). Kernel of weights (left, bottom). Generated value (right, bottom).

3.3.1 Programming environment

The application part of the thesis has been written with the help of the **Emacs** text editor, mainly because it provides a uniformity which supports various working flows, since it is not focused on a specific programming language, like PyCharm or Eclipse, for example. The editor is highly customizable and it provides structured, powerful commands for text editing, file searching, in-place code versioning and many more.

3.3.2 Programming language

For the purpose of the thesis we have used the **Python** programming language mainly because of its numerous available libraries which support computer vision, heavy mathematical computations and raster image processing. The most important libraries used in this application are listed at Section 3.3.3.

3.3.3 Libraries

NumPy

Since the data used for this thesis is large in size, relying on Python alone for data structures for processing does not scale. Python is an interpreted language, not a compiled one, which means that for each operation its interpreter has to do extra work such that it translated the bytecode instruction into a form that is machine executable. For a large number of operation, such as we have high resolution images, this does not scale. Instead, we resorted to using NumPy, which is an open-source library which brings support of large data computation, such as multi-dimensional arrays and matrices, as well as a large collection of mathematical functions which can be easily used for their operation [CRH]. The library has a **well optimised C code core** which can handle large processing by bypassing the python interpreter, which results in the speed of a compiled language, not an interpreted one. Therefore, based on the needs of our dataset, we have used NumPys' main data structure, the ndarray (n-dimensional array) as a data structure to hold our images and various optimisations applied throughout processing focused on using only this type of data structure in order to make use of NumPys' optimisations and speed.

Open Source Computer Vision Library (OpenCV)

The OpenCV library brings support for **computer vision processing**, by providing programming functions which can be reliably used [cud]. Its use in our thesis both for image alignment and detecting motion through deep optical flow, by using its already integrated functions as a support. OpenCV, as well as NumPy, is open-source and it is written and optimised in native code, which is needed for the large size of our dataset entities.

Geospatial Data Abstraction Library (GDAL)

The GDAL library is a translator for vector and **raster geo-spatial data formats**, which presents a single data model to work with [gda]. Since our images are represented in the tiff format, reading them by using GDAL proved to be the easy and reliable.

Version-control

In order to keep track of various changes, bugs and enhancements of our application, we had to make sure that we are using a **reliable version-control** system. For this purpose, Git was chosen for its fast performance with the aim of data integrity and support for distributed workflows [git]. When a directory is flagged as a git repository, it keeps track of any changes made to that location, independent of a central server of network accessibility. By keeping a clean workflow, one can easily include enhancements and try experiments without the worry of losing stability. We have kept track of every change, bug and enhancement from the beginning of developing this application by using git, and the repository which contains all of this information can be found on Github, at `https://github.com/BabyCakes13/GlacierImagePredictor`.

Chapter 4

Performance and experiments

Here we should also write on what machine I ran the code and try it on different ones as a thing. Also write about the idea of moving the processing on cloud, but there might be traffic overhead because the images are large. Also say that we store it on harddrive, NOT ssd. Make tests with both.

Chapter 5

Conclusions

5.1 Future Development

During the process of creating the application, multiple **problems** were met. One of them was that even though there are a large number of aerial images collected in the collection provided for the Landsat 8 satellite, there are not many which have a small enough cloud coverage. This results in datasets with a low number of entities, which means that we cannot test our results on a longer period of time. Since clouds interfere with delimiting snow and pixels from a scene, if we wanted to broaden our dataset, we thought of two options:

- harvesting images from **multiple satellites**, such as Sentinel and older versions of Landsat. Even if these satellites use other types of sensors on board for Earth observation, by matching the wavelengths of their bands one could make a match between them. By using multiple satellites, more populated datasets can be created, since we are taking into consideration collected scenes over the years for pairs of paths, rows and months. By only using Landsat 8, even with an allowed cloud coverage of 20%, the average length of a $path, row, month$) dataset was around 15;
- implementing **cloud mapping algorithms** such that cloud could be detected and trimmed out before the calculation of the normalized snow difference index and optical flow extraction. For now, the optical flow algorithm ran between scenes which have a high cloud coverage finds large vectors of motion at the locations where clouds existed from one frame to another. The result does not take into

consideration the difference between snow pixels and cloud ones, therefore all of them are moved.

Another improvement could be changing the way that the coordinates of the pixels of the motion generated image are calculated. For now, we are predicting their location by using the method described in 2.15, where we are multiplying by two both the original coordinates and the time. A more interesting approach would be using **time series forecasting models** applied on the distance vectors generated by optical flow for each pixel over time in order to generate future entries. This could be done with statistical methods such as the autoregressive integrated moving average (ARIMA). However, doing this forecast over each pixel of a high resolution image would take a lot of processing power; also, a larger $path, row, month$ dataset would be needed for such an analysis.

One of the main slow downs in developing and using the application was the **slow time of processing** due to the large data files. As described in Section 4, running the NDSI calculation, alignment, optical flow generation and motion image creation took around two minutes for each scene. By having access to a more powerful machine, this time could be reduced. This could be done by **migrating** the processing unit to **cloud** and hiring a machine with a CPU which has a high number of cores and better performance. One of the possible downsides would then be the amount of time needed for passing the large **scene files** through he network to the machine on cloud, which, depending on the **bandwidth**. A solution to this downside would be passing the dataset before starting the processing and make sure that they are on the same machine. However, cloud storage can get very expensive and the trade off between processing time and **expenses** could prove too big.

As for a final future development idea, migrating the codebase from Python to a compiled programming language, such as C or C++ would yield in much faster results.

Bibliography

[bd] bd. `https://landsat.gsfc.nasa.gov/landsat-8/landsat-8-bands/`. Accessed: 2021-06-22.

[CMZ] Jin Xiao Huanchao Du Chaoqun Ma, Xiaoguang Hu and Guofeng Zhang.

[CRH] [...] Travis E. Oliphant Charles R. Harris, K. Jarrod Millman.

[cud] Cuda.

[Der10] Konstantinos G. Derpanis. Overview of the ransac algorithm. 2010.

[ERB] Kurt Konolige Ethan Rublee, Vincent Rabaud and Gary Bradski.

[FB81] M.A. Fischler and R.C. Bolles. Random sample consensus: A paradigm for modelfitting with applications to image analysis and automated cartography. 1981.

[gda] Gdal.

[git] git.

[Hal15] Dr. Dorothy K Hall. Viirs snow cover algorithm theoretical basis document (atbd). 2015.

[HY15] Xianjian Lu Hongbo Yan, Guoqing Zhou. Comparative analysis of surface soil moisture retrieval using vswi and tvdi in karst areas. In *Proceedings Volume 9808, International Conference on Intelligent Earth Observing and Applications 2015; 980806 (2015)*, 2015.

[KK14] Edward J Knight and Geir Kvaran. Landsat-8 operational land imager design, characterization and performance. In *Proceedings of the ECAI 94 Workshop*

on Applications of Evolutionary Algorithms. Landsat-8 Sensor Characterization and Calibration, 2014.

[l8o] l8otb. https://gisgeography.com/landsat-8-bands-combinations/. Accessed: 2021-06-22.

[LANa] Landsat. https://www.usgs.gov/core-science-systems/nli/landsat/landsat-8?qt-science_support_page_related_con=0#. Accessed: 2021-06-21.

[LANb] Landsatpic. https://landsat.gsfc.nasa.gov/sites/landsat/files/2013/01. Accessed: 2021-06-21.

[lc1] lc1l1. https://prd-wret.s3.us-west-2.amazonaws.com/assets/palladium/production/atoms/files/LSDS-1656_%20Landsat_Collection1_L1_Product_Definition-v2.pdf. Accessed: 2021-06-22.

[lgn] lgng. https://landsat.gsfc.nasa.gov/. Accessed: 2021-06-21.

[LK81] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. 1981.

[MKI] Jan-Gunnar Winther Max K6nig and Elisabeth Isaksson.

[nds] ndsi. https://nsidc.org/support/faq/what-ndsi-snow-cover-and-how-does-it-comp Accessed: 2021-06-22.

[opt] opticalflow. https://nanonets.com/blog/optical-flow/. Accessed: 2021-06-25.

[orb] orb. https://docs.opencv.org/4.5.2/d1/d89/tutorial_py_orb.html. Accessed: 2021-06-24.

[RRA19] Adina E. Racoviteanu, Karl Rittger, and Richard Armstrong. An automated approach for estimating snowline altitudes in the karakoram and eastern himalaya from remote sensing. 2019.

[SHWS18] Christopher Nuth1 Liss M. Andreassen2 Ward J. J. van Pelt Solveig H. Winsvold, Andreas Kääb1 and Thomas Schellenberger1. Using sar satellite data time series for regional glacier mapping. 2018.

[sn] sn. https://gisgeography.com/landsat-file-naming-convention/. Accessed: 2021-06-22.

[STA] Stac. https://stacspec.org/STAC-api.html. Accessed: 2021-06-20.

[TK20] Swati Tak and Ashok K. Keshari. Investigating mass balance of parvati glacier in himalaya using satellite imagery based model. 2020.

[USG] Usgs. https://earthexplorer.usgs.gov/. Accessed: 2021-06-21.

[VA] Vinay S. Shekhara Akshay KumarC. K N Balasubramanya Murthya S. Natarajanb Vinay A., Avani S. Rao.

[WGI] World glacier inventory. http://nsidc.org/data/glacier_inventory/index.html. Accessed: 2021-06-20.

[WKN16] Solveig Havstad Winsvold, Andreas Kääb, and Christopher Nuth. Regional glacier mapping using optical satellite data time series. 2016.

[wrs] wrs. https://landsat.gsfc.nasa.gov/about/worldwide-reference-system/. Accessed: 2021-06-22.

Chapter 6

Glossary

6.1 Acronyms

WGI	World Glacier Inventory
NDSI	Normalized Snow Difference Index
CSV	Comma separated values
JSON	JavaScript Object Notation

Table 6.1: Acronyms table